CSE 331:
Algorithms & Complexity

“Shortest Path”

Prof. Charlie Anne Carlson (She/Her)

Lecture 17

Wednesday October 10^{th}, 2025

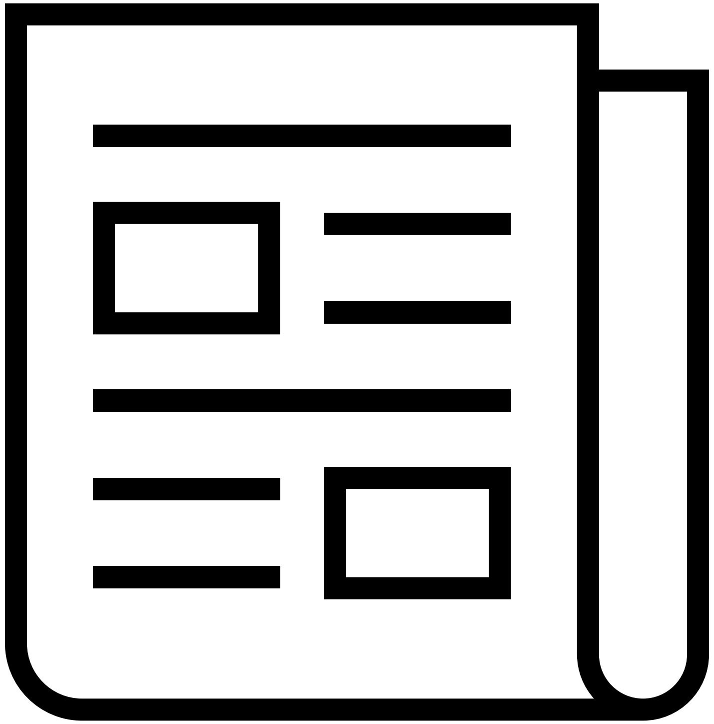
University at Buffalo

Schedule

1. Course Updates
2. Interval Scheduling
3. Stay Ahead
4. Runtime Analysis
5. Shortest Path

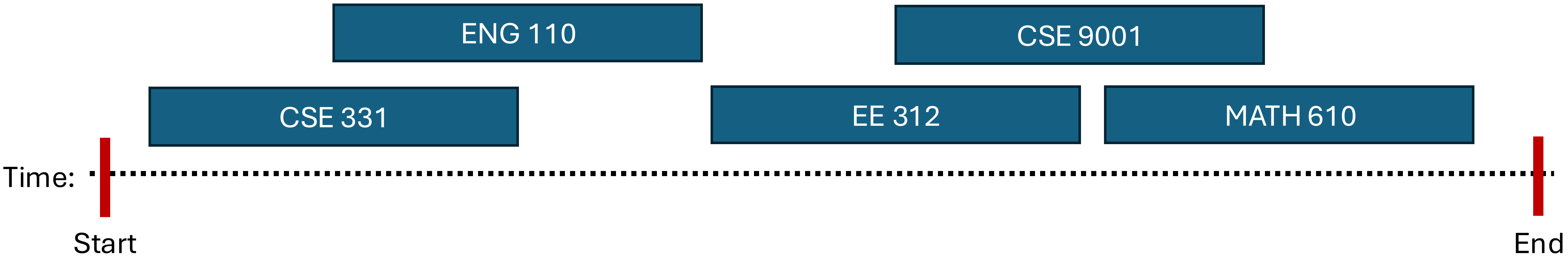
Course Updates

- All Grading Before Tuesday
- HW 4 Out
  - Not Due Next Week!
- Group Project
  - First Problems Oct 31st



Interval Scheduling

- Consider an interval of time (e.g. Wednesday).
- Consider tasks that need to be completed during specific times (e.g. classes).
- We want to fit as many tasks as possible into the day such that no two overlap.



Finish First Algorithm

- **Input**: List of n tasks R
  - For each $i \in R$, let $s(i)$ and $f(i)$ be start and finish times.
- **Output**: List of non-conflicting tasks of maximum length
  - Let S be empty
  - While R is not empty:
    - Find $\mathrm{i} \in R$ with earliest finish time (argmin $\mathrm{f(i)}$)
    - Add i to S
    - Remove all tasks that conflict with i from R
  - Return S

Claim: The Finish First Algorithm is Optimal

Proof Ideas:

- Let S be the set returned by the algorithm.
  - Let $i_1, i_2, \dots, i_k$ be the elements in S sorted by finish times.
- Let S^* be the optimal list.
  - Let $j_1, j_2, \dots, j_m$ be the elements in S^* sorted by finish times.
- We want to show that for every index $1 \leq \ell \leq k$, $f(i_\ell) \leq f(j_\ell)$.

Claim: The Finish First Algorithm is Optimal

Proof Ideas:

- We want to show that for every index $1 \leq \ell \leq \mathrm{k}$, $f(i_\ell) \leq \mathrm{f}(\mathrm{j}_\ell)$.
  - That is, we want to show that our algorithm is always doing better than the optimal solution! "Stay Ahead"
- If this is true, then it must be the case that $\mathrm{m} = \mathrm{k}$.

Claim: The Finish First Algorithm is Optimal

Proof Ideas:

- We want to show that for every index $1 \leq \ell \leq \mathrm{m}$, $f(i_\ell) \leq \mathrm{f}(\mathrm{j}_\ell)$.
  - That is, we want to show that our algorithm is always doing better than the optimal solution! “Stay Ahead”
- If this is true, then it must be the case that $\mathrm{m} = \mathrm{k}$.
  - If not, then our algorithm would have added $j_{\mathrm{k}+1}$ to S.

Claim: The Finish First Algorithm is Optimal

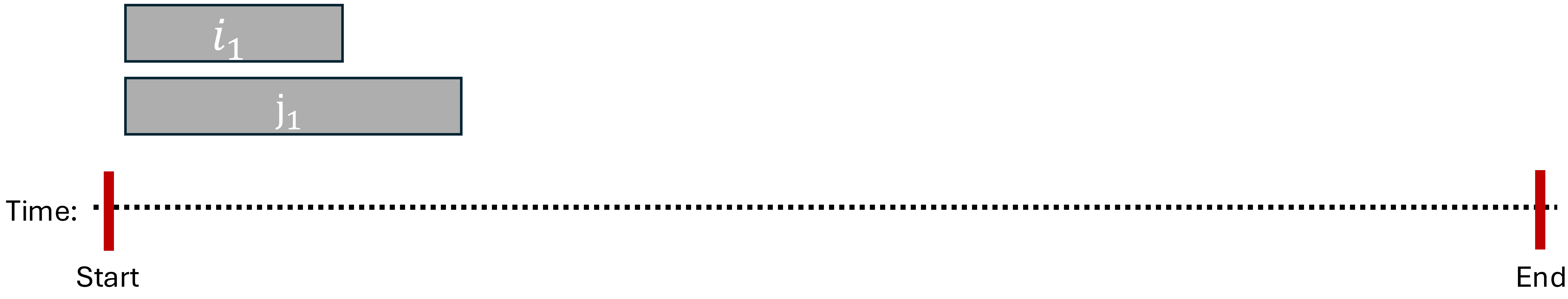
Proof Ideas:

- We want to show that for every index $1 \leq \ell \leq \mathrm{m}$, $f(i_\ell) \leq \mathrm{f}(\mathrm{j}_\ell)$.
  - That is, we want to show that our algorithm is always doing better than the optimal solution! “Stay Ahead”
- If this is true, then it must be the case that $\mathrm{m} = \mathrm{k}$.
  - If not, then our algorithm would have added $j_{\mathrm{k}+1}$ to S.

Claim: The Finish First Algorithm is Optimal

Base Case:

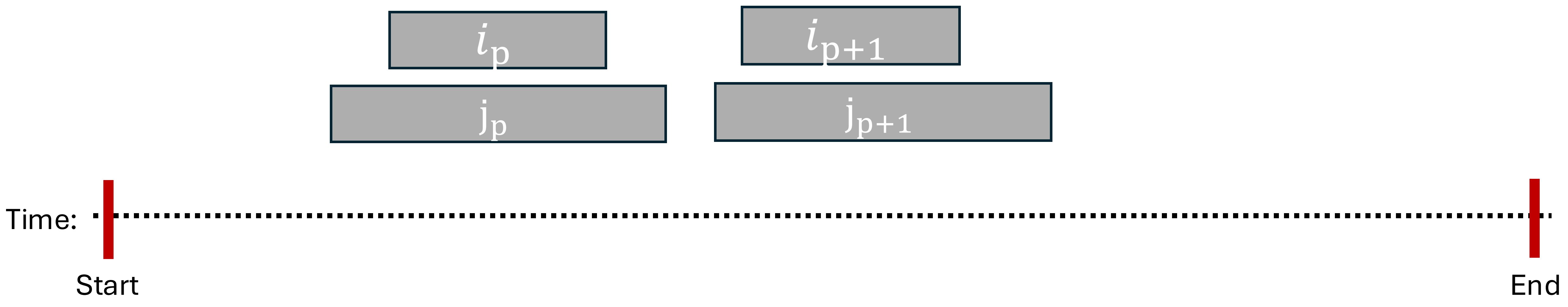
- We observe that $f(i_1) \leq f(j_1)$ since the algorithm always takes the element with the quickest finish time.



Claim: The Finish First Algorithm is Optimal

Inductive Hypothesis:

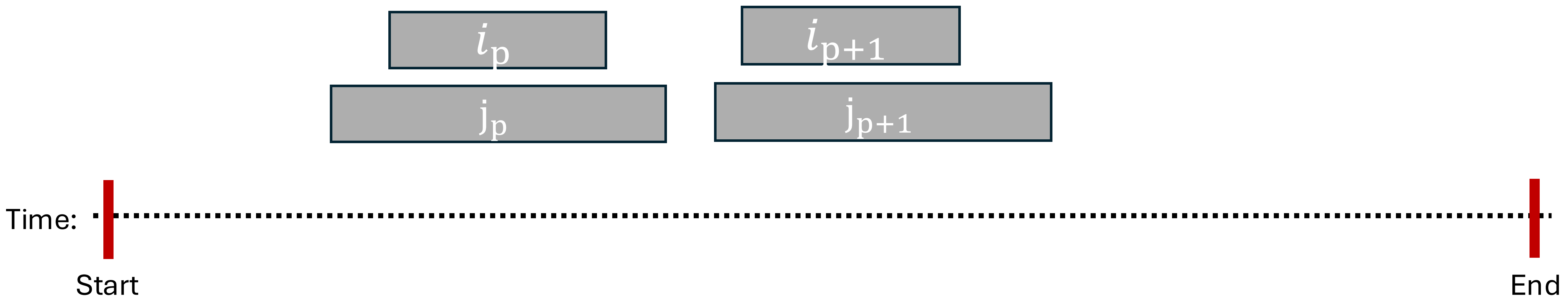
- Assume that $f(i_{\mathrm{p}}) \le f(j_{\mathrm{p}})$ for some $1 \le \mathrm{p} < \mathrm{k}$.
  - We will prove that $f(i_{\mathrm{p+1}}) \le f(j_{\mathrm{p+1}})$



Claim: The Finish First Algorithm is Optimal

Inductive Case:

- Observe that since $f(i_{\mathrm{p}}) \leq f(j_{\mathrm{p}})$, $j_{\mathrm{p+1}}$ was in the set R when we added $i_{\mathrm{p+1}}$.
  - Hence, $f(i_{\mathrm{p+1}}) \leq f(j_{\mathrm{p+1}})$



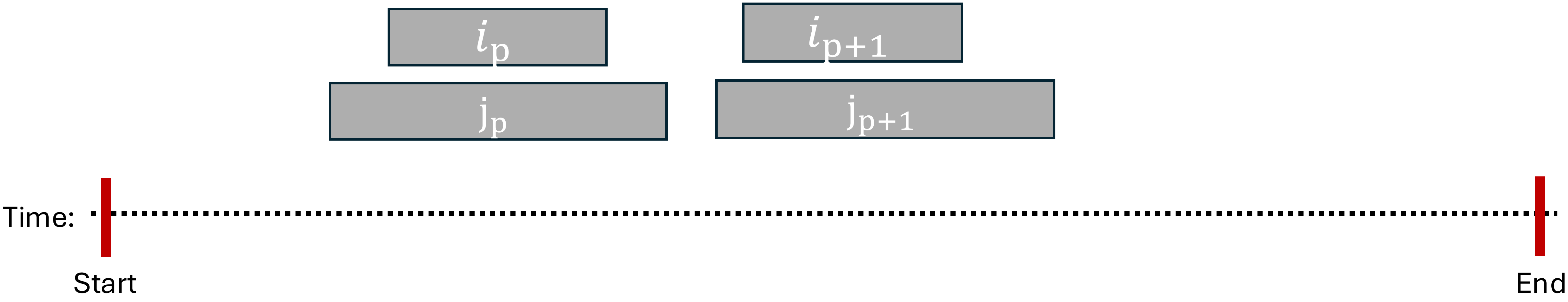
Finish First Algorithm

- **Input**: List of n tasks R
  - For each $i \in R$, let $s(i)$ and $f(i)$ be start and finish times.
- **Output**: List of non-conflicting tasks of maximum length
  - Let S be empty
  - While R is not empty:
    - **Find $\boldsymbol{i} \in \boldsymbol{R}$ with earliest finish time (argmin $\boldsymbol{f(i)}$)**
    - Add i to S
    - Remove all tasks that conflict with i from R
  - Return S

Claim: The Finish First Algorithm is Optimal

Conclusion:

- We have shown $f(i_\ell) \leq f(j_\ell)$ for all $1 \leq \ell \leq \mathrm{k}$ as desired.
- If $\mathrm{k} \leq \mathrm{m}$, then we could add $\mathrm{j}_{\mathrm{k}+1}$ to S which contradicts the while loop exit condition.



Finish First Algorithm

- **Input**: List of n tasks R
  - For each $i \in R$, let $s(i)$ and $f(i)$ be start and finish times.
- **Output**: List of non-conflicting tasks of maximum length
  - Let S be empty
  - **While $\boldsymbol{R}$ is not empty**:
    - Find $i \in R$ with earliest finish time (argmin $f(i)$)
    - Add i to S
    - Remove all tasks that conflict with i from R
  - Return S

Runtime

- **Input**: List of n tasks R
  - For each $i \in R$, let $s(i)$ and $f(i)$ be start and finish times.
- **Output**: List of non-conflicting tasks of maximum length
  - Let S be empty
  - While R is not empty:
    - Find $i \in R$ with earliest finish time (argmin $f(i)$)
    - Add i to S
    - Remove all tasks that conflict with i from R
  - Return S

Runtime

- **Input**: List of n tasks R
  - For each $i \in R$, let $s(i)$ and $f(i)$ be start and finish times.
- **Output**: List of non-conflicting tasks of maximum length
  - Let S be empty
  - While R is not empty:
    - Find $i \in R$ with earliest finish time (argmin $f(i)$)
    - Add i to S
    - Remove all tasks that conflict with i from R
  - Return S

These look like O(n) steps!

Runtime

- **Input**: List of n tasks R
  - For each $i \in R$, let $s(i)$ and $f(i)$ be start and finish times.
- **Output**: List of non-conflicting tasks of maximum length
  - Let S be empty
  - Sort R by finish time
  - Let last_finished = 0
  - For i ∈ [n]:
    - If s(i) ≥ last_finished:
      - Add i to S
      - Set last_finished = f(i)
  - Return S

Because we sorted, this is next task to finish that won't conflict!

Runtime

- **Input**: List of n tasks R
  - For each $i \in R$, let $s(i)$ and $f(i)$ be start and finish times.
- **Output**: List of non-conflicting tasks of maximum length
  - Let S be empty
  - Sort R by finish time ← This only takes O(nlog(n)) time!
  - Let last_finished = 0
  - For i ∈ $[n]$:
    - If $s(i) \geq$ last_finished:
      - Add i to S
      - Set last_finished = $f(i)$
  - Return S

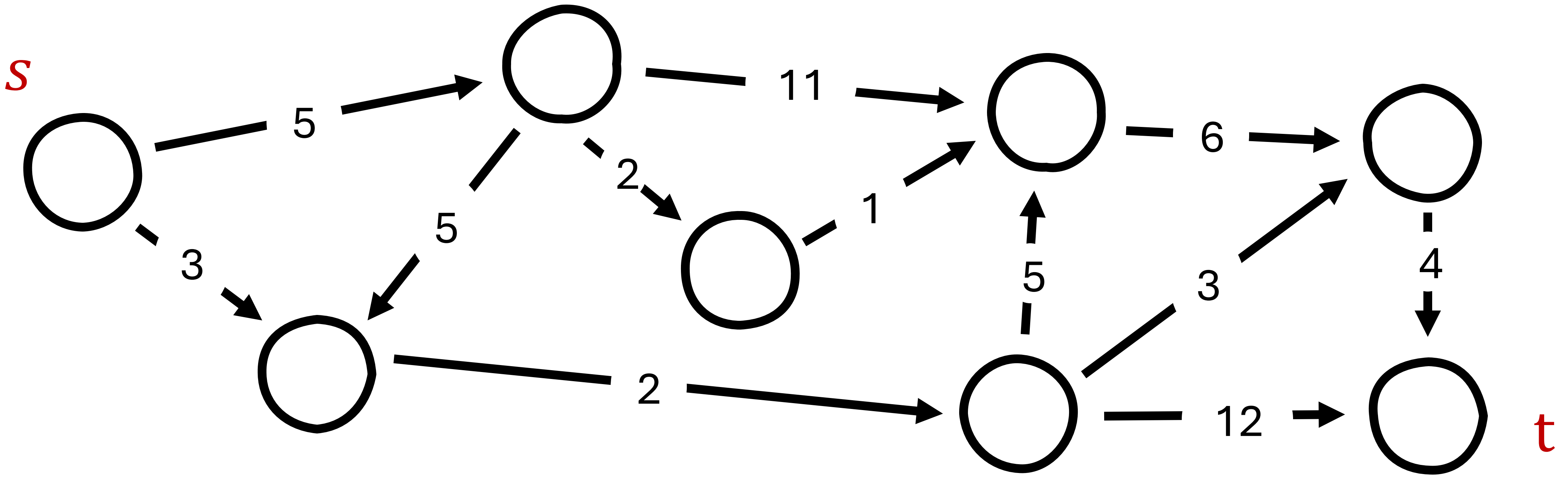
Runtime O(nlog(n))

- **Input**: List of n tasks R
  - For each $i \in R$, let $s(i)$ and $f(i)$ be start and finish times.
- **Output**: List of non-conflicting tasks of maximum length
  - Let S be empty
  - Sort R by finish time
  - Let last_finished = 0
  - For i $\in$ [n]:
    - If s(i) $\geq$ last_finished:
      - Add i to S
      - Set last_finished = f(i)
  - Return S

Each step in this loop is constant time!

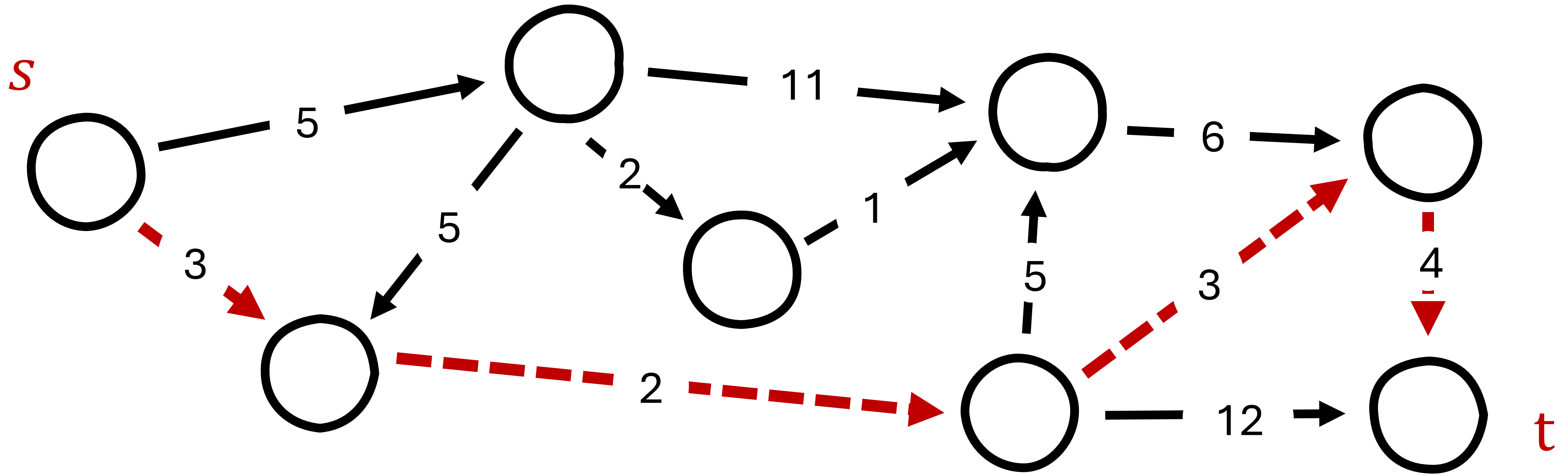
Shortest Path

- **Input**: Directed graph $G = (V, E)$, edge lengths $\ell: E \to R_{\geq 0}$, a source vertex $s \in V$, and a destination vertex $t \in V$.
- **Output**: Find the shortest directed path from s to t in G.



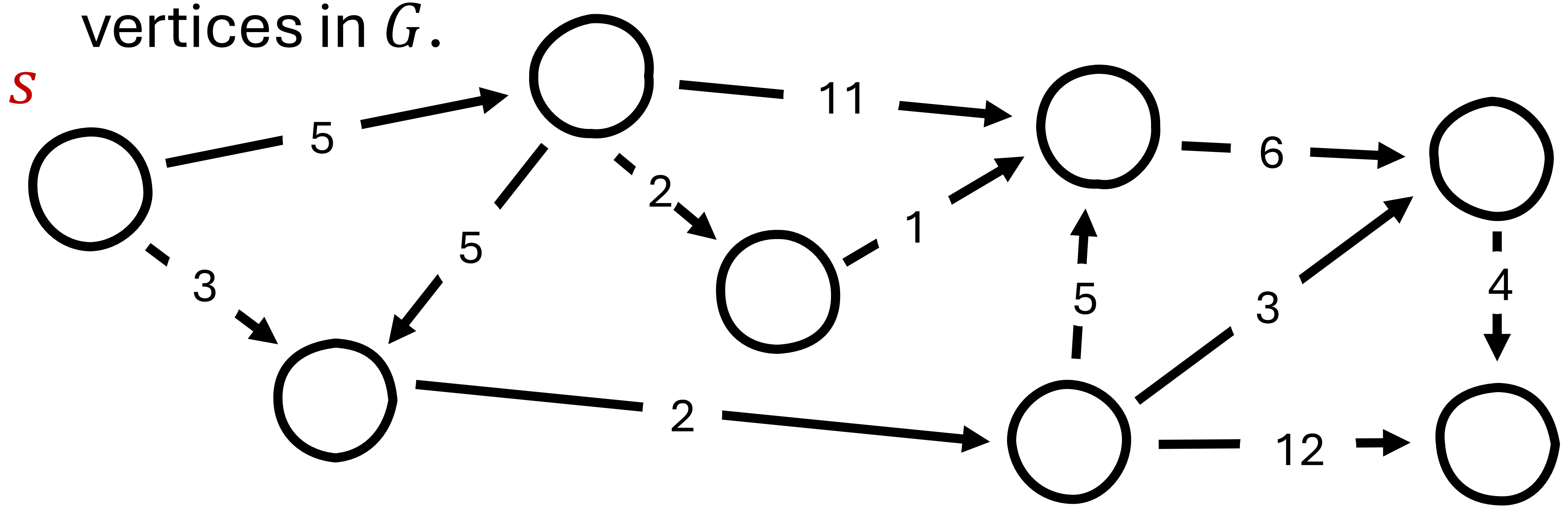
Single Pair Shortest Path

- **Input**: Directed graph $G = (V, E)$, edge lengths $\ell: E \to R_{\geq 0}$, a source vertex $s \in V$, and a destination vertex $t \in V$.
- **Output**: Find the shortest directed path from s to t in G.



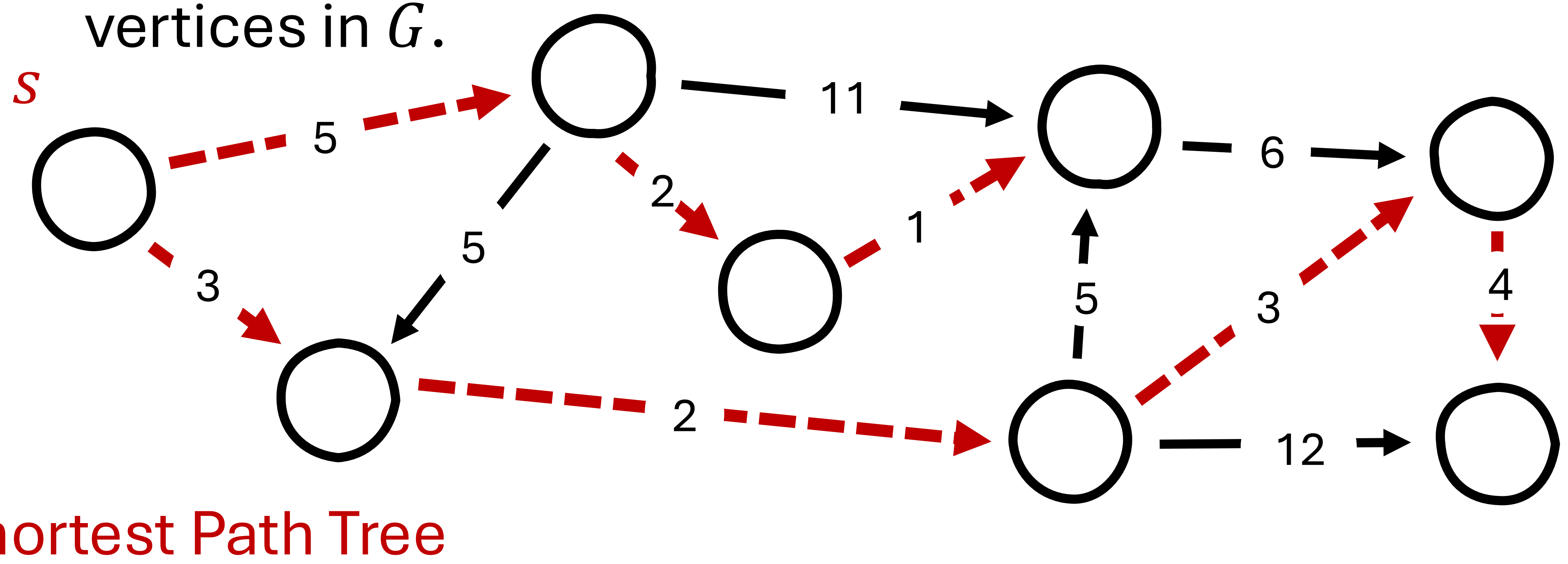
Single Pair Shortest Path

- **Input**: Directed graph $G = (V, E)$, edge lengths $\ell: E \to R_{\geq 0}$, and a source vertex $s \in V$.
- **Output**: Find the shortest directed path from s to all vertices in G.



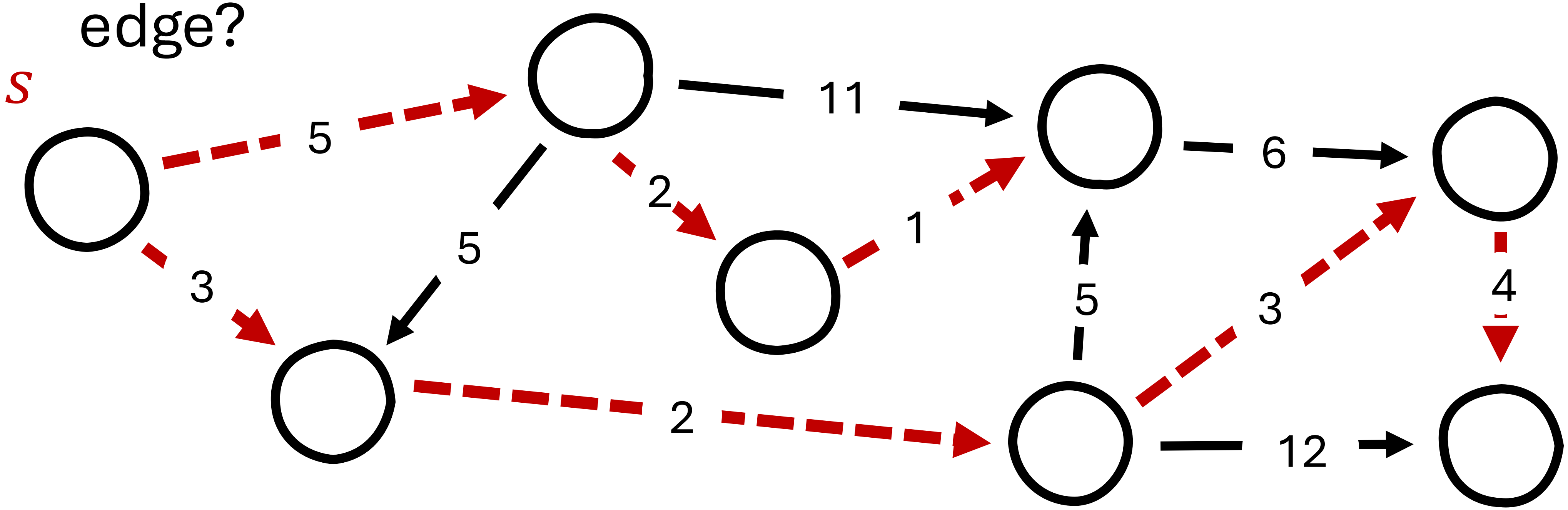
Single Pair Shortest Path

- **Input**: Directed graph $G = (V, E)$, edge lengths $\ell: E \rightarrow R_{\geq 0}$, and a source vertex $s \in V$.
- **Output**: Find the shortest directed path from s to all vertices in G.



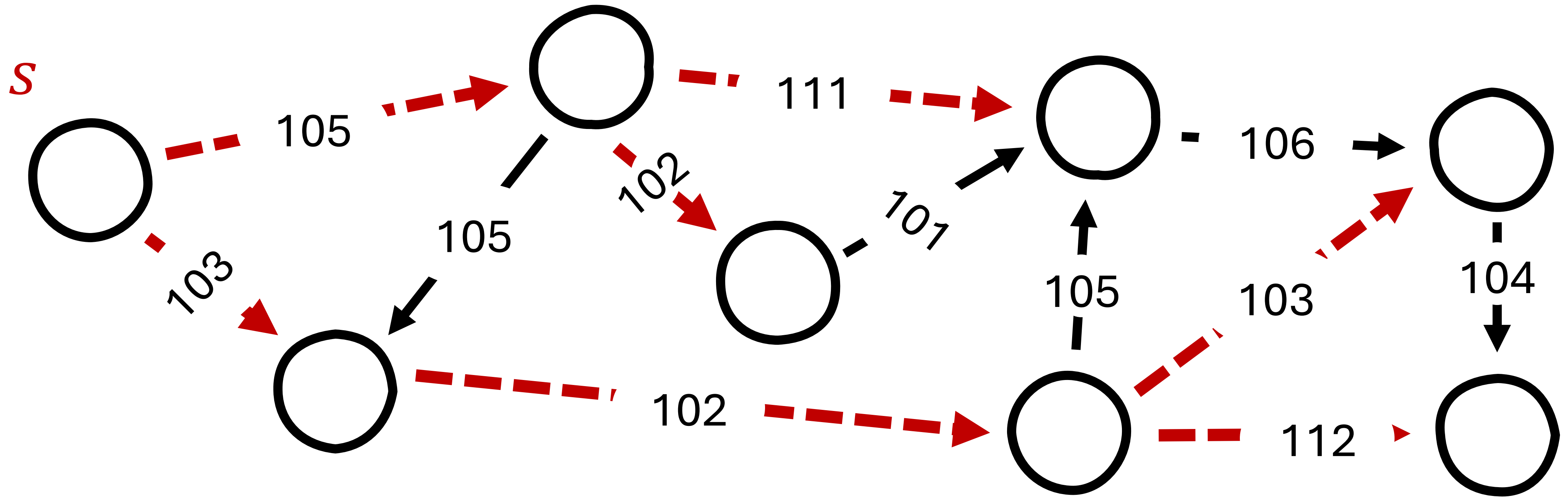
Questions

- Q1: How do you solve one problem with the answer to the other?
- Q2: What happens to the tree when you add 100 to each edge?



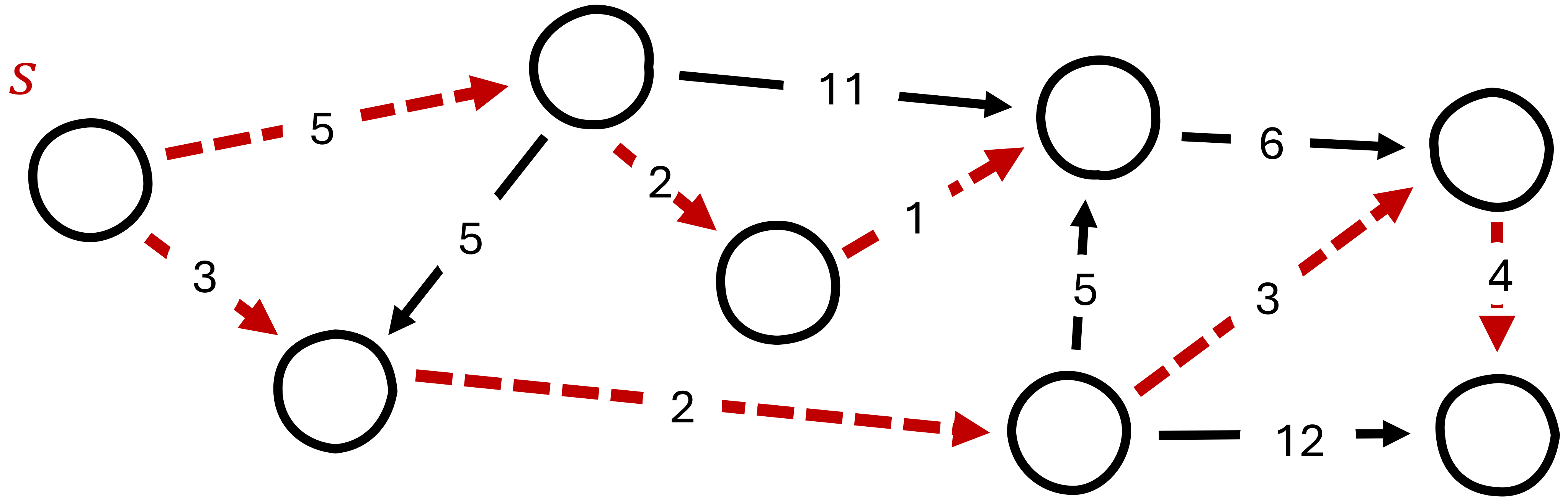
Questions

- A1: You can lookup t or run with t for each vertex.)
- A2: The shortest paths may change!



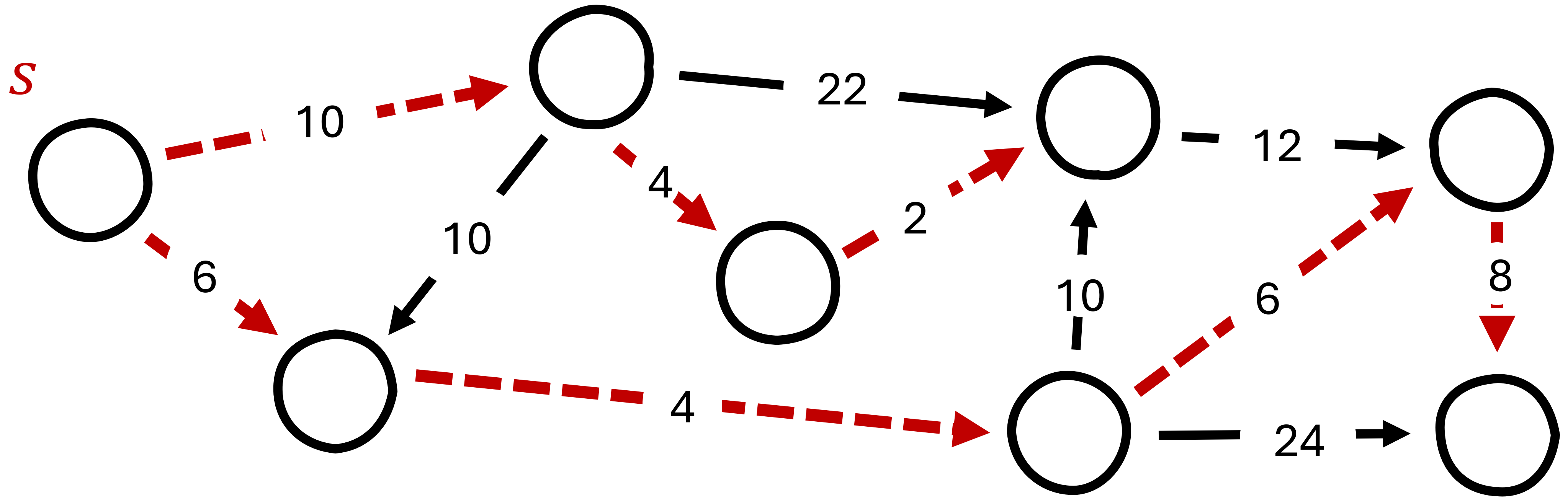
Questions

- Q3: How can you solve this if each edge has length 1?
- Q2: What happens to the tree when you multiply each length by 2?



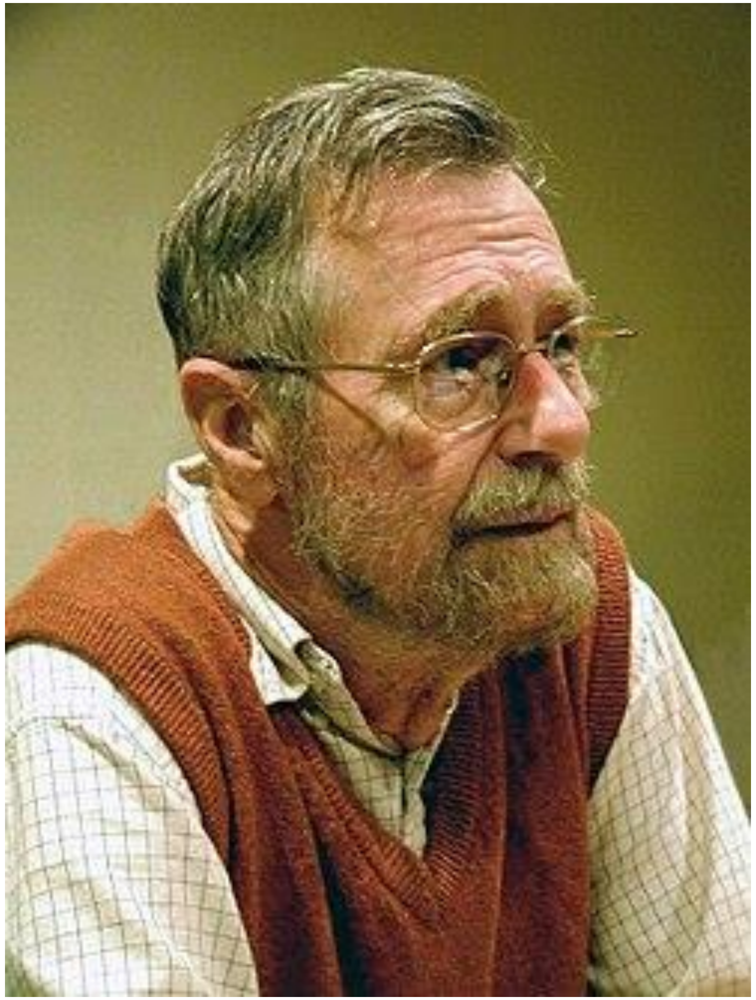
Questions

- Q3: You can run BFS.
- Q2: The shortest path is still the shortest path but twice as long.



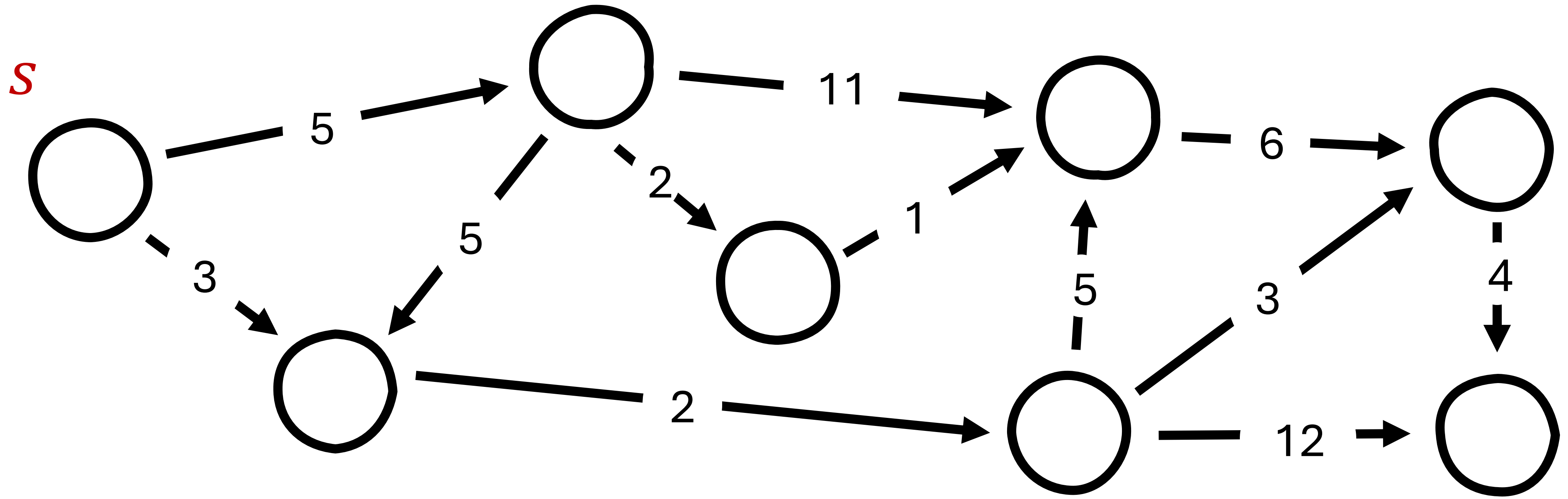
Dijkstra's Greedy Algorithm

- We will describe a **greedy algorithm** that is named after its discoverer, Edsger Wybe Dijkstra. ->
- The algorithm runs in O(|E| + |V|log(|V|)) time when implemented with a **priority queue**.
- We will assume no negative edge weights.



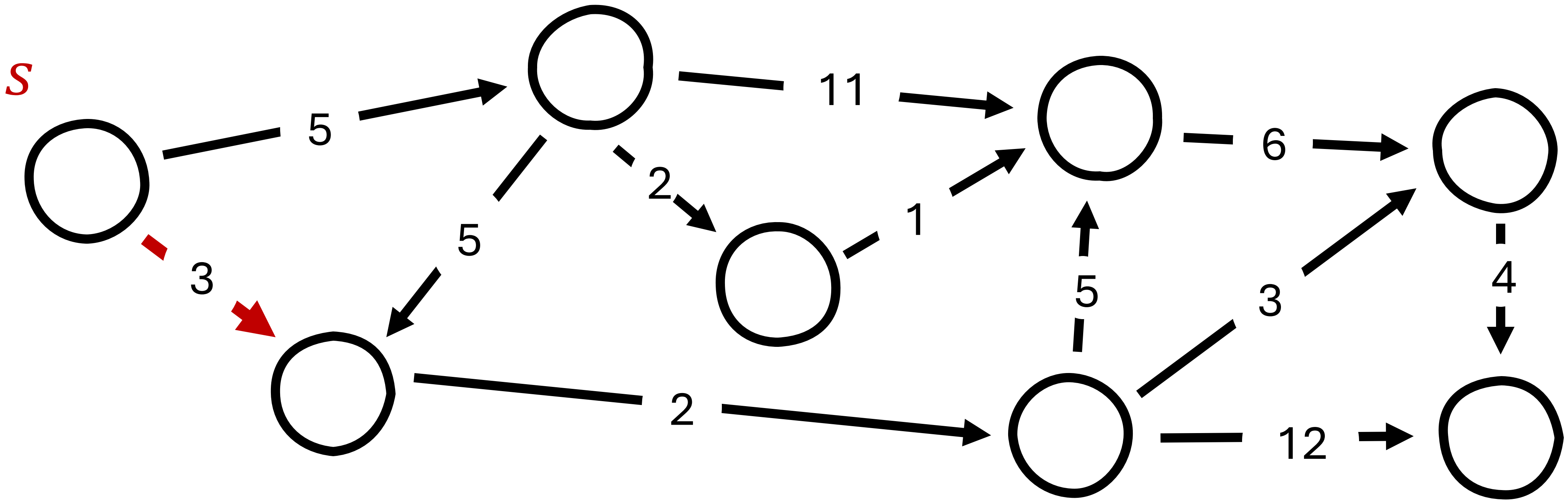
Dijkstra’s Algorithm Idea

- Prompt: What edge do we always know is going to be part of a shortest path?



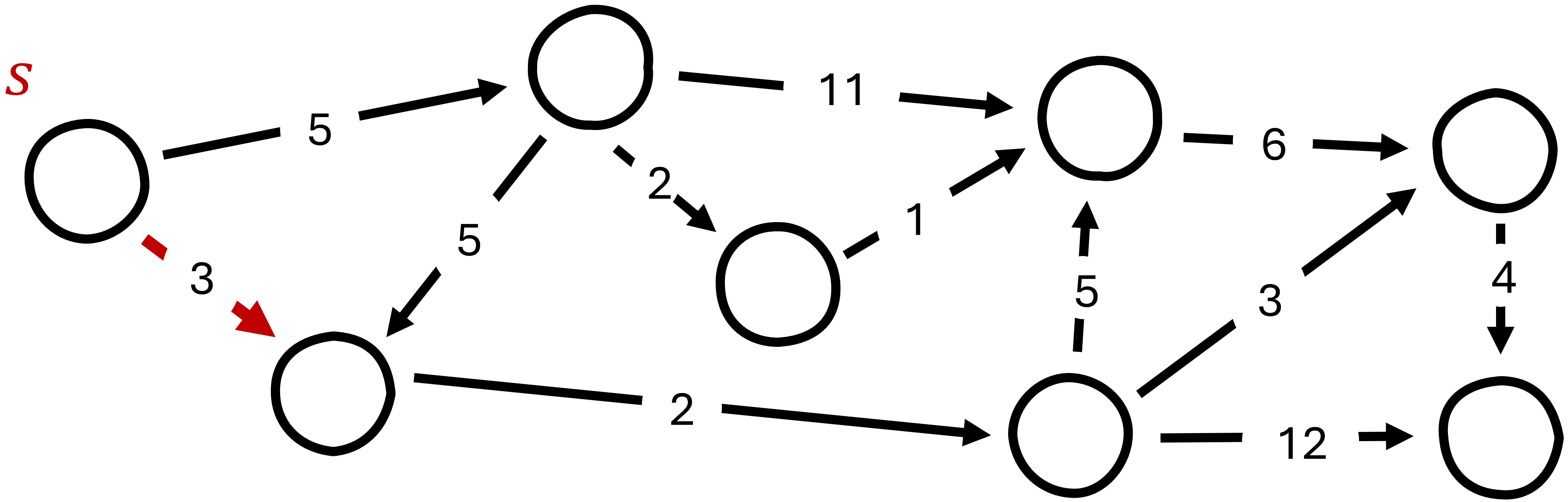
Dijkstra's Algorithm Idea

- Observation: The shortest edge leaving the source vertex is always in a shortest path.



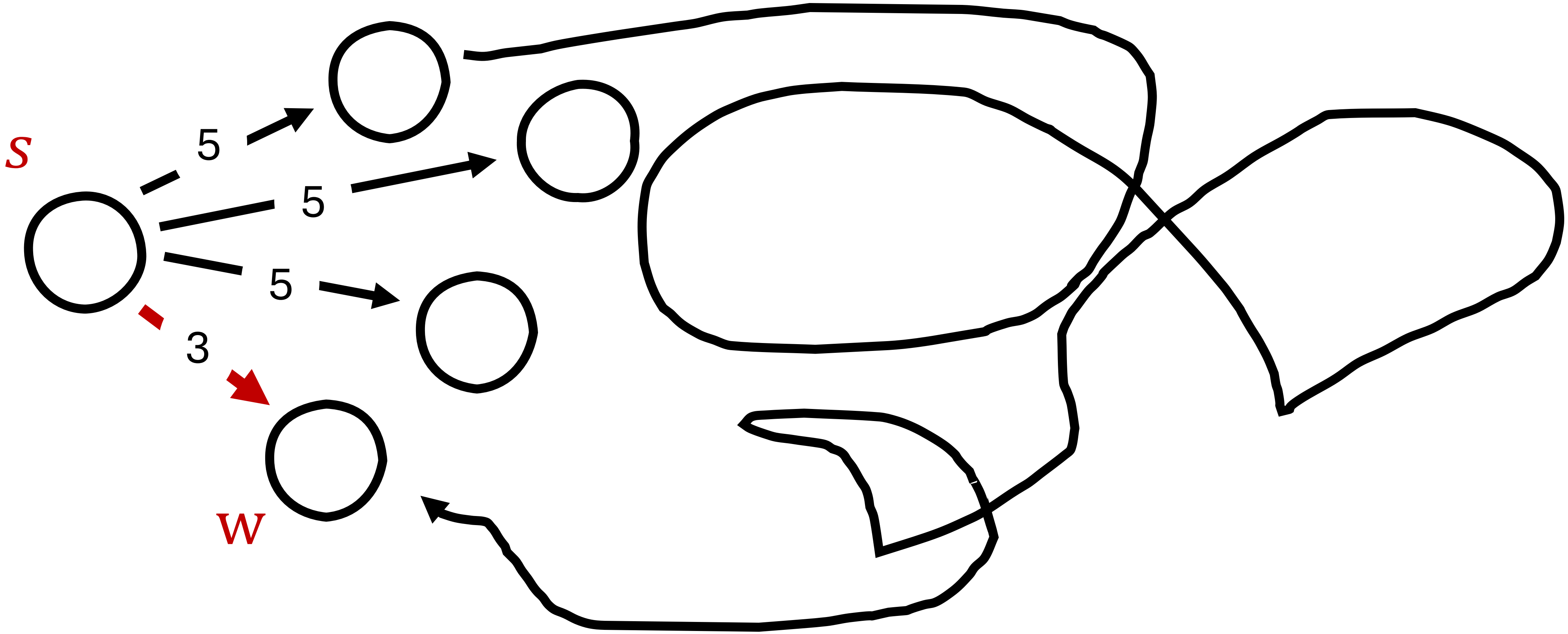
Dijkstra's Algorithm Idea

- Motivation: You know that the shortest path from where you are to anywhere else is important.



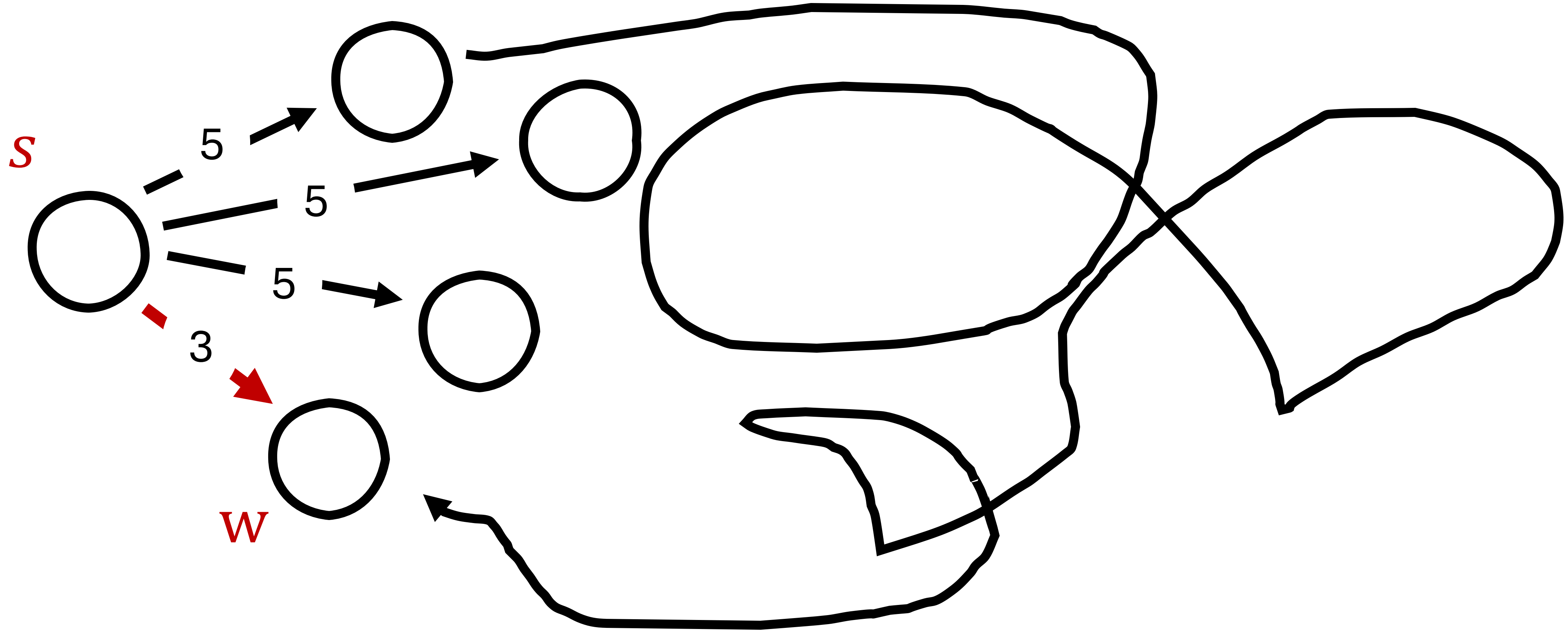
Dijkstra’s Algorithm Idea

- Proof: Suppose the shortest edge leaving s went to w. Now suppose there is another path from s to w. It must use another edge leaving s.



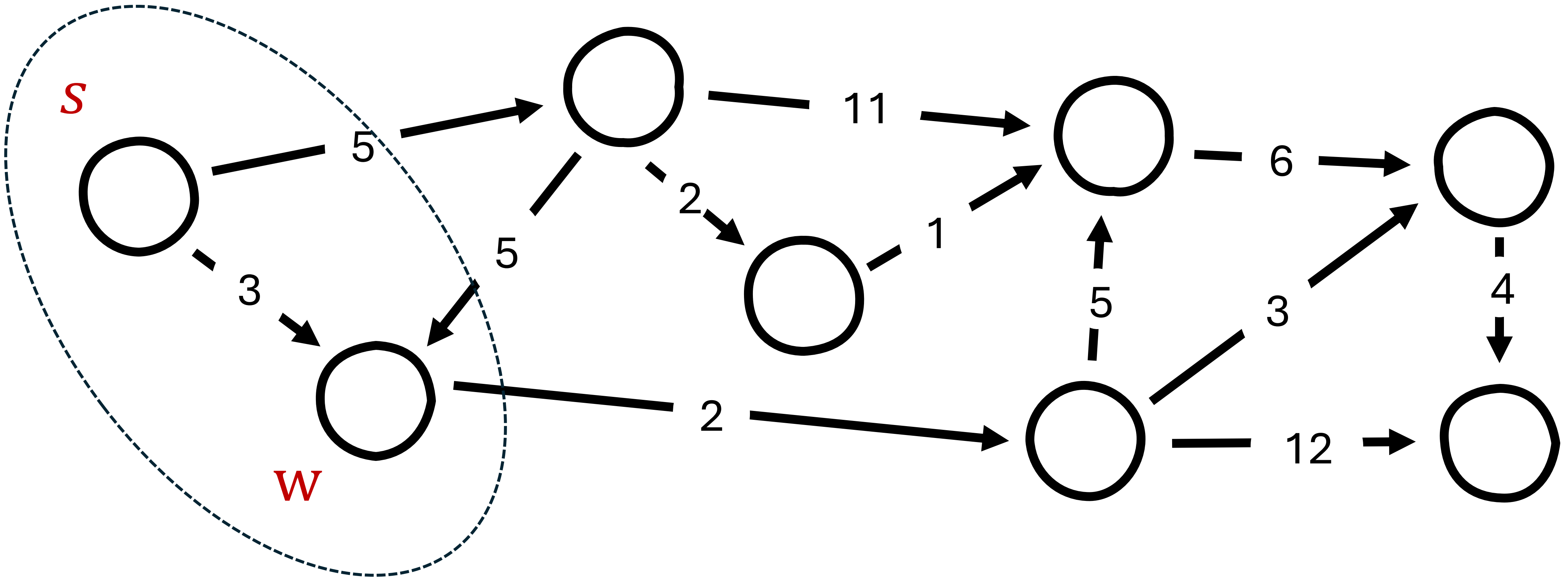
Dijkstra’s Algorithm Idea

- If it uses another edge leaving s then it must be just as long or longer of a path!



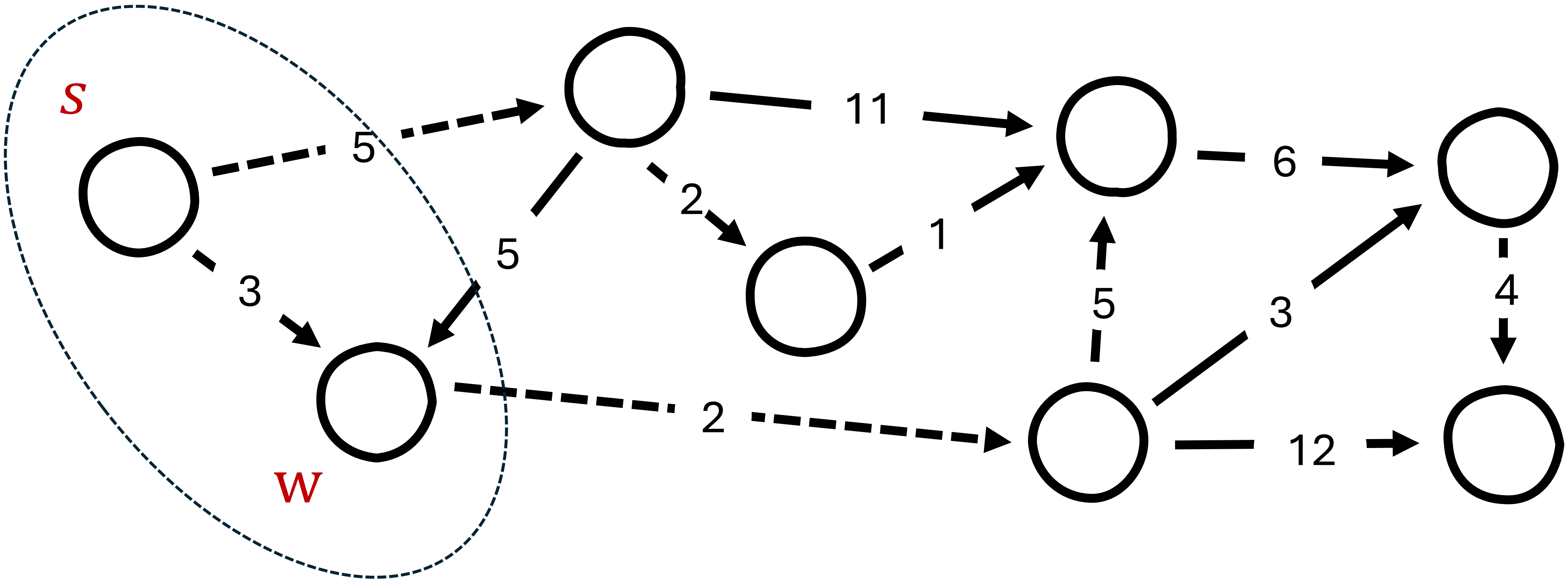
Dijkstra’s Algorithm Idea

- Prompt: What can we say about the shortest path tree with respect to s and w?



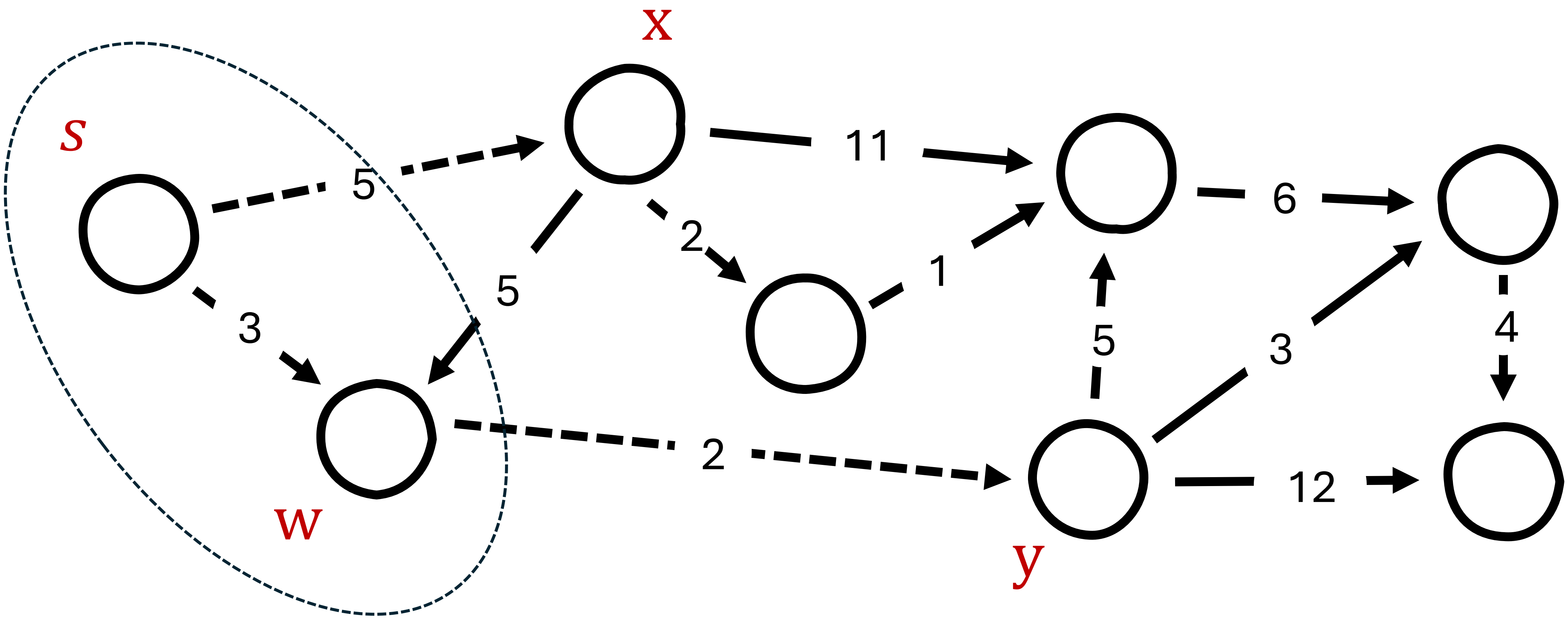
Dijkstra's Algorithm Idea

- Observation: The shortest path tree must include an edge leaving s and/or w. Otherwise, you can't reach everything.



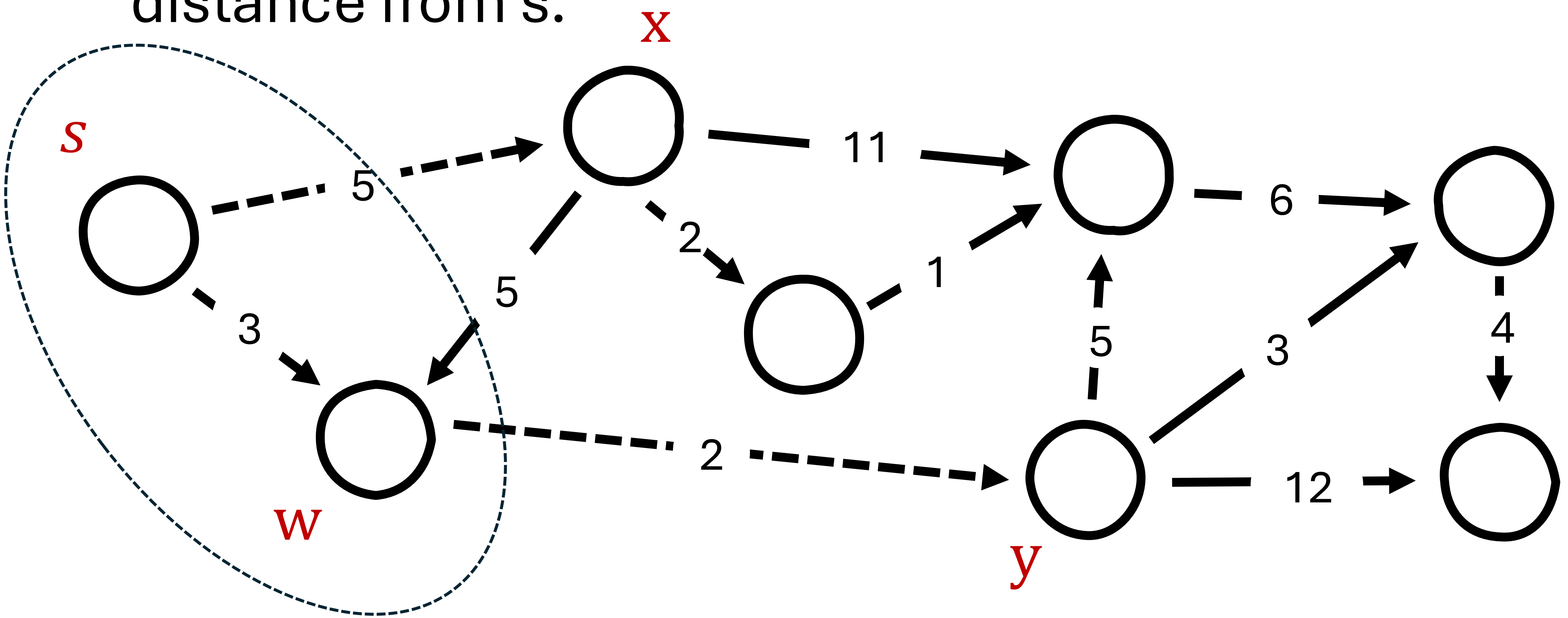
Dijkstra's Algorithm Idea

- Prompt: Which is closer to s?



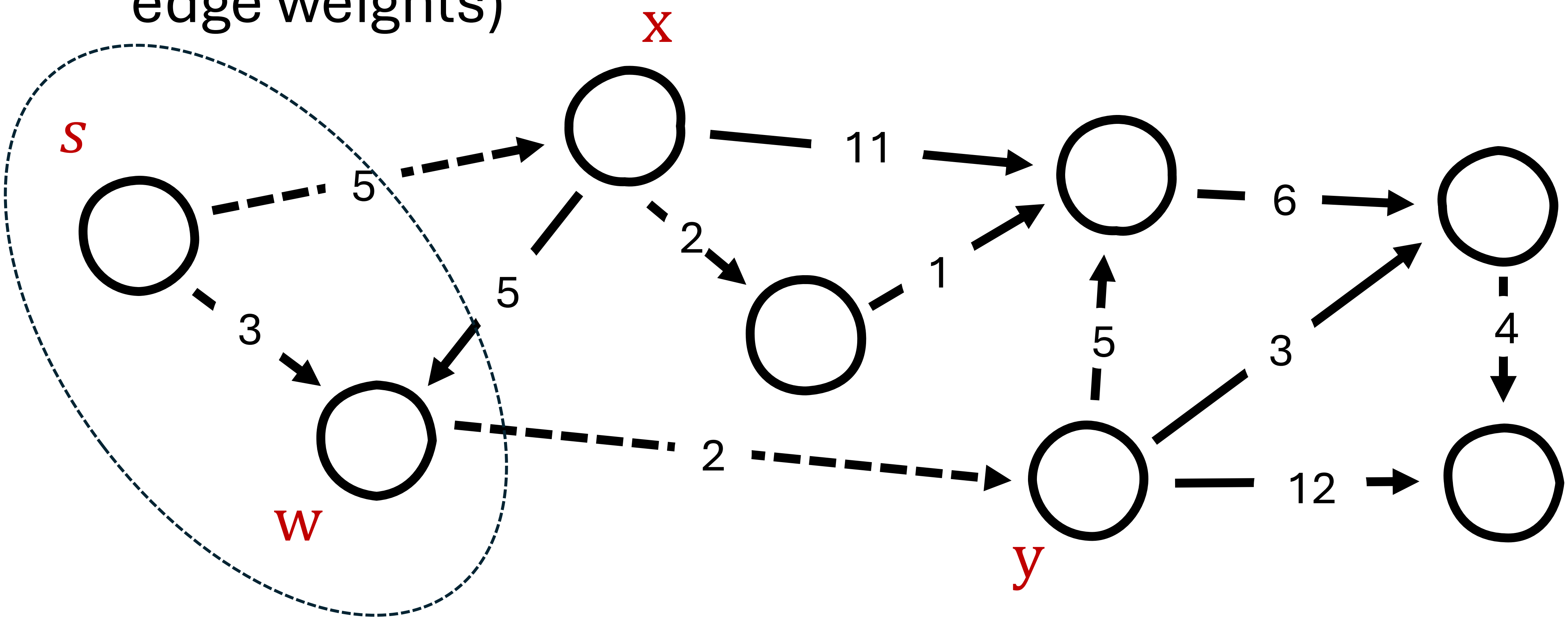
Dijkstra's Algorithm Idea

- Observation: There are both 5 away from s. We don't want to assume that the distance from w is the same as the distance from s.



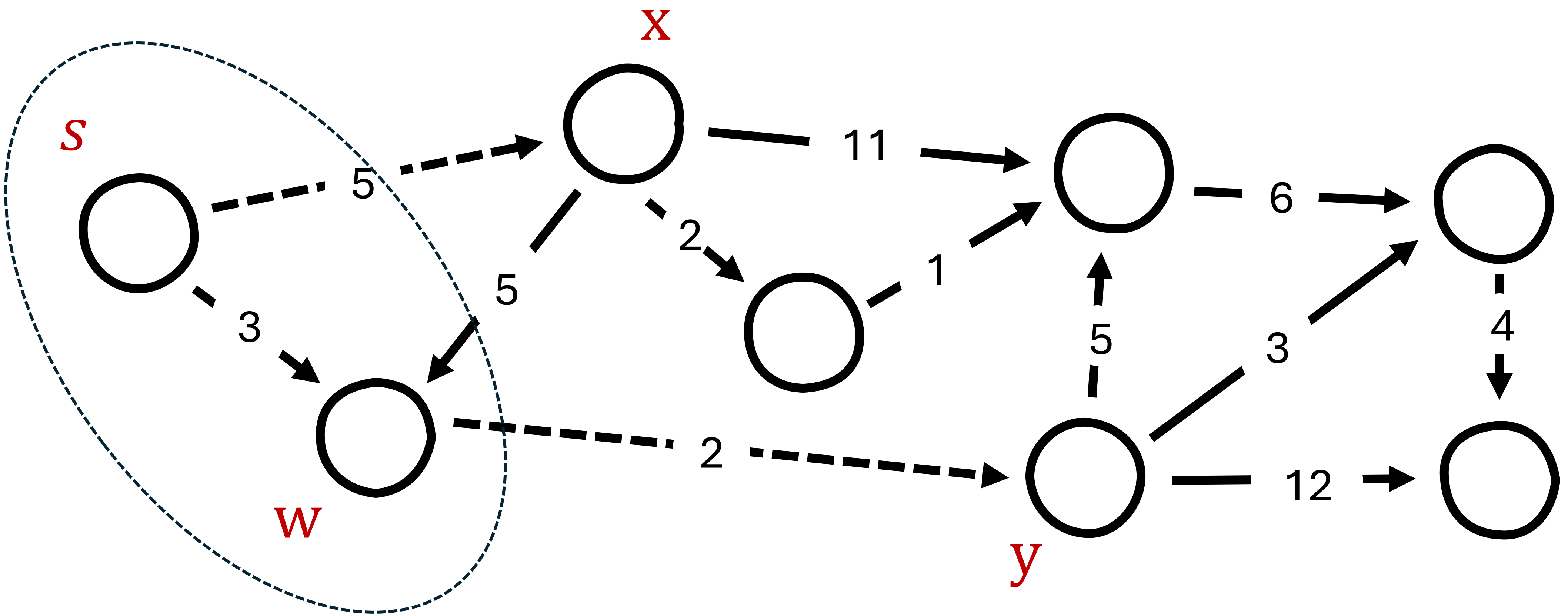
Dijkstra’s Algorithm Idea

- Observation: The s to x edge and the w to y edge should both be in the shortest path tree. (We need non-negative edge weights)



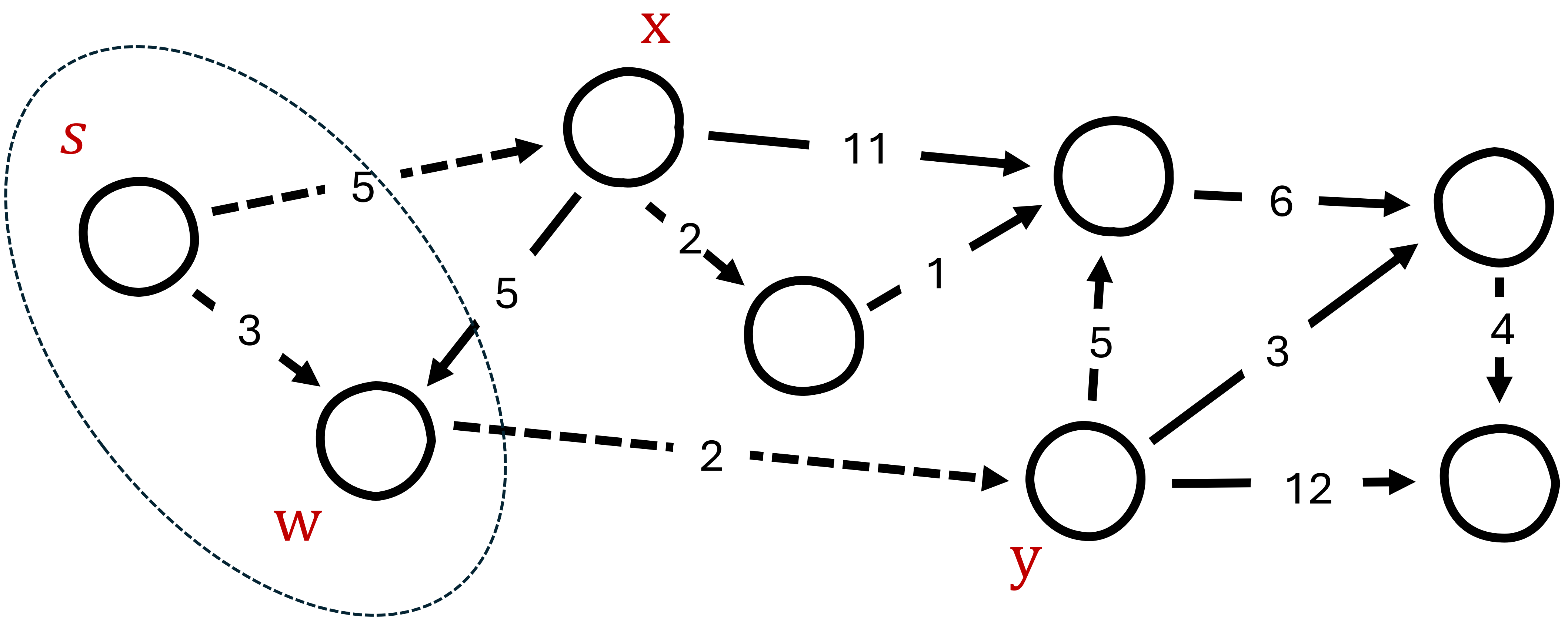
Dijkstra's Algorithm Idea

- Prompt: How do we generalize these observation as a greedy rule?



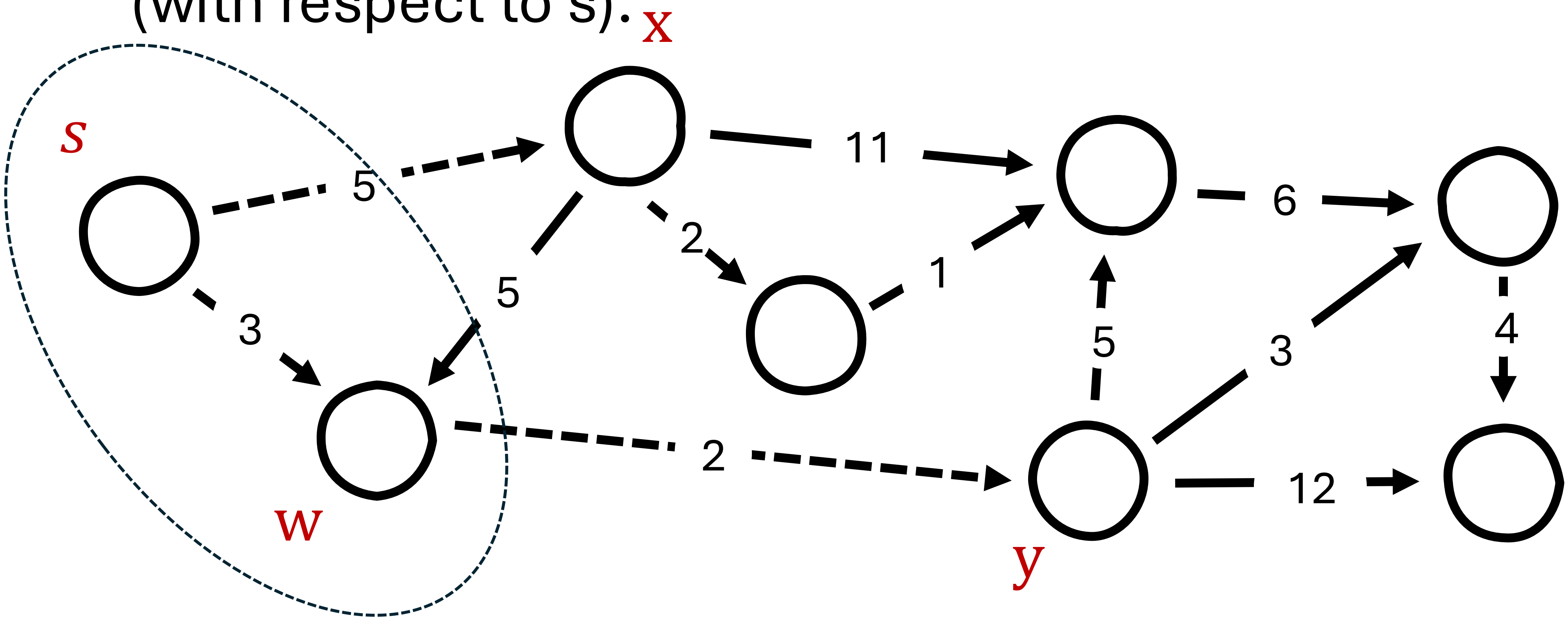
Dijkstra's Algorithm Idea

- Algorithm Idea: In each iteration, pick an edge from my shortest path tree to the "nearest" vertex not yet in the tree.



Dijkstra's Algorithm Idea

- Algorithm Idea: Keep track of the shortest path tree. In each iteration, add an edge from the tree to the nearest neighbor (with respect to s).



Dijkstra's Algorithm

Dijkstra's Algorithm (G, ℓ)

Let S be the set of explored nodes

  For each $u \in S$, we store a distance $d(u)$

Initially $S = \{s\}$ and $d(s) = 0$

While $S \neq V$

  Select a node $v \notin S$ with at least one edge from S for which

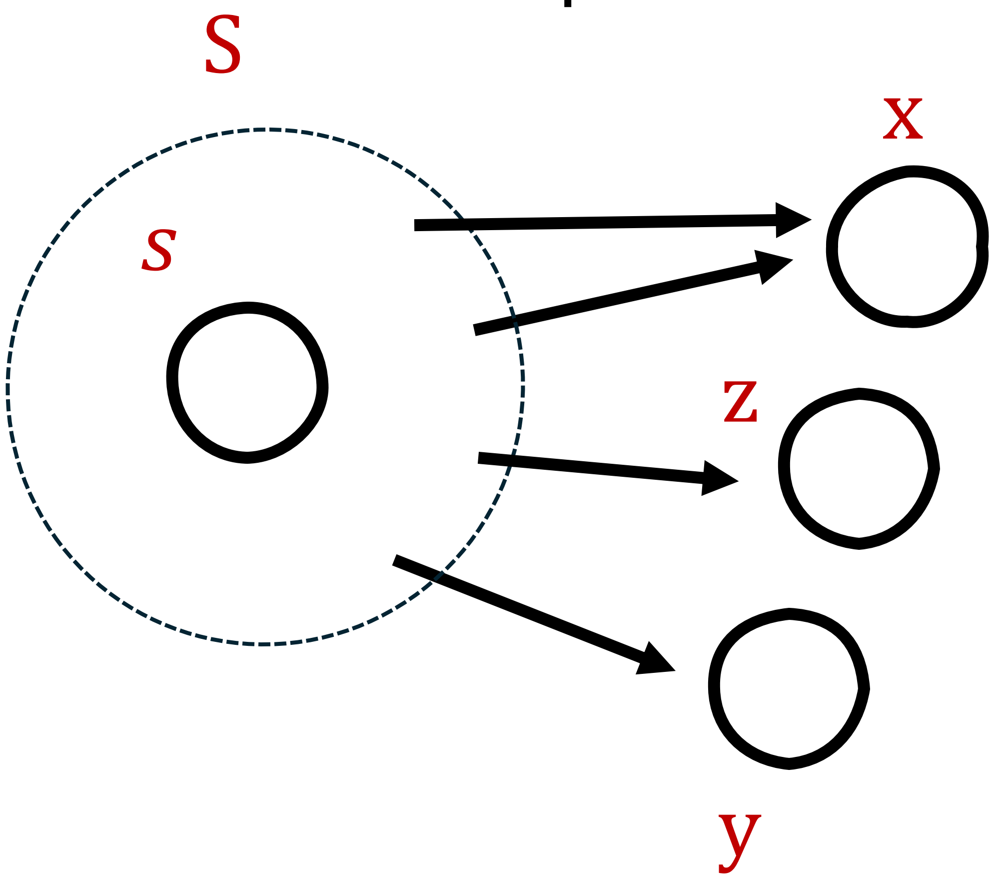
    $d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible

  Add v to S and define $d(v) = d'(v)$

EndWhile

Dijkstra's Algorithm Idea

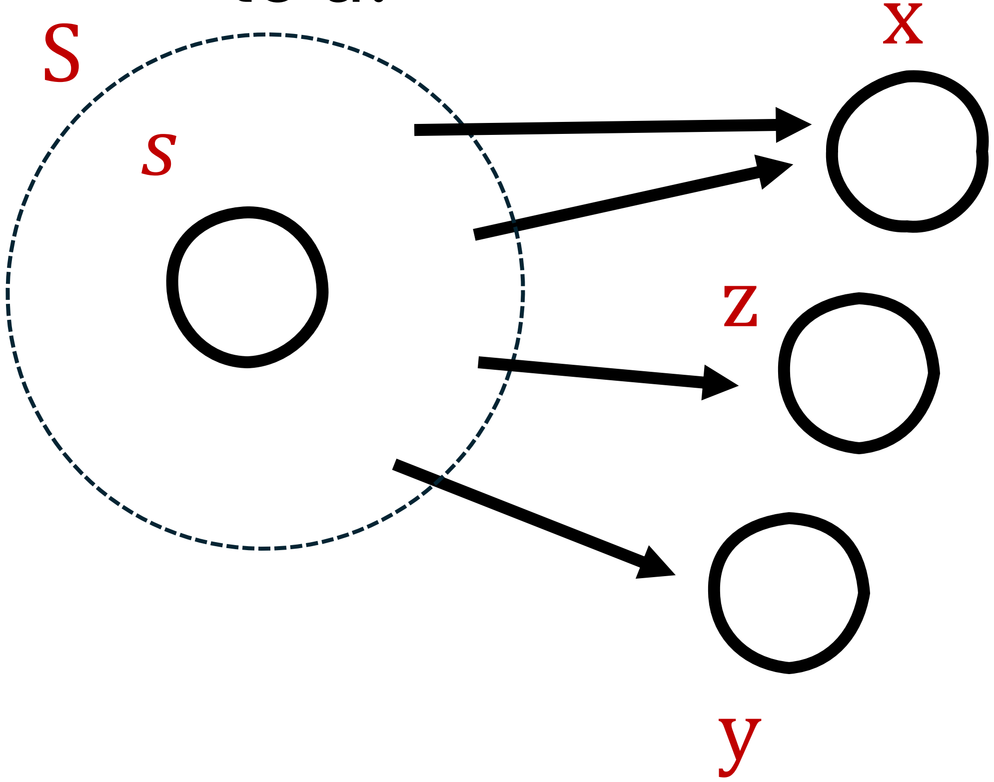
- Look at all neighbors of S. Determine which has the shortest path from s. Add that to S. Repeat.




```
Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each u ∈ S, we store a distance d(u)
Initially S = {s} and d(s) = 0
While S ≠ V
    Select a node v ∉ S with at least one edge from S for which
        d'(v) = min_{e=(u,v):u∈S} d(u) + ℓ_e is as small as possible
    Add v to S and define d(v) = d'(v)
EndWhile
```

Dijkstra's Algorithm Idea

- Key Idea: We guess the distance from s to a neighbor is $min_{e=(u,v):u\in S} d(u) + \ell_e$ where $d(u)$ is the distance from s to u.



Dijkstra's Algorithm (G, ℓ)

Let S be the set of explored nodes

  For each $u \in S$, we store a distance $d(u)$

Initially $S = \{s\}$ and $d(s) = 0$

While $S \neq V$

  Select a node $v \notin S$ with at least one edge from S for which

    $d'(v) = \min_{e=(u,v):u\in S} d(u) + \ell_e$ is as small as possible

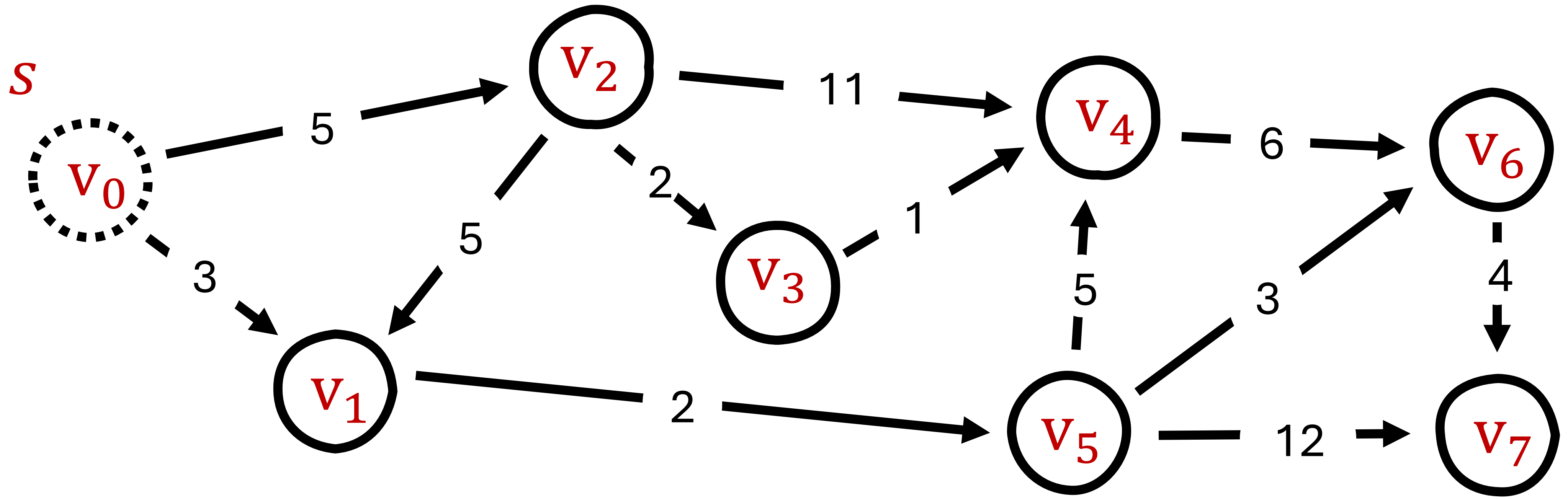
  Add v to S and define $d(v) = d'(v)$

EndWhile

Dijkstra's Algorithm

S = {s}
d = [0,∞, ∞, ∞, ∞, ∞, ∞, ∞]

Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each $u \in S$, we store a distance $d(u)$
Initially $S = \{s\}$ and $d(s) = 0$
While $S \neq V$
    Select a node $v \notin S$ with at least one edge from S for which
        $d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible
    Add v to S and define $d(v) = d'(v)$
EndWhile

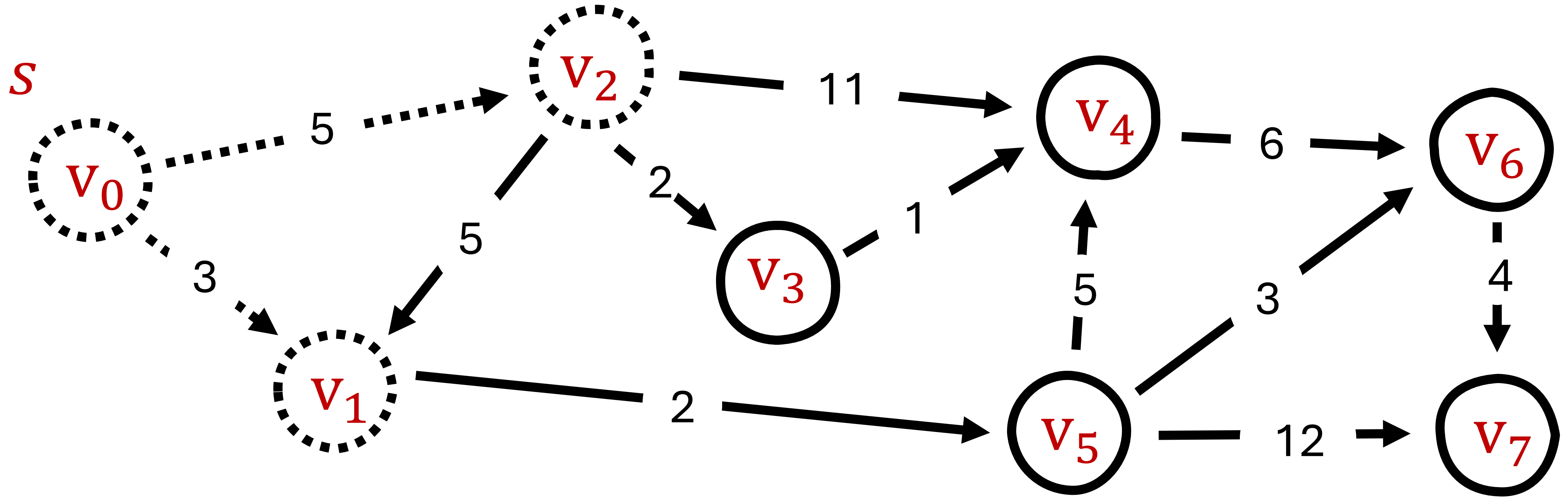


Dijkstra’s Algorithm

S = {s}

d = [0, 3, 5, ∞, ∞, ∞, ∞, ∞]

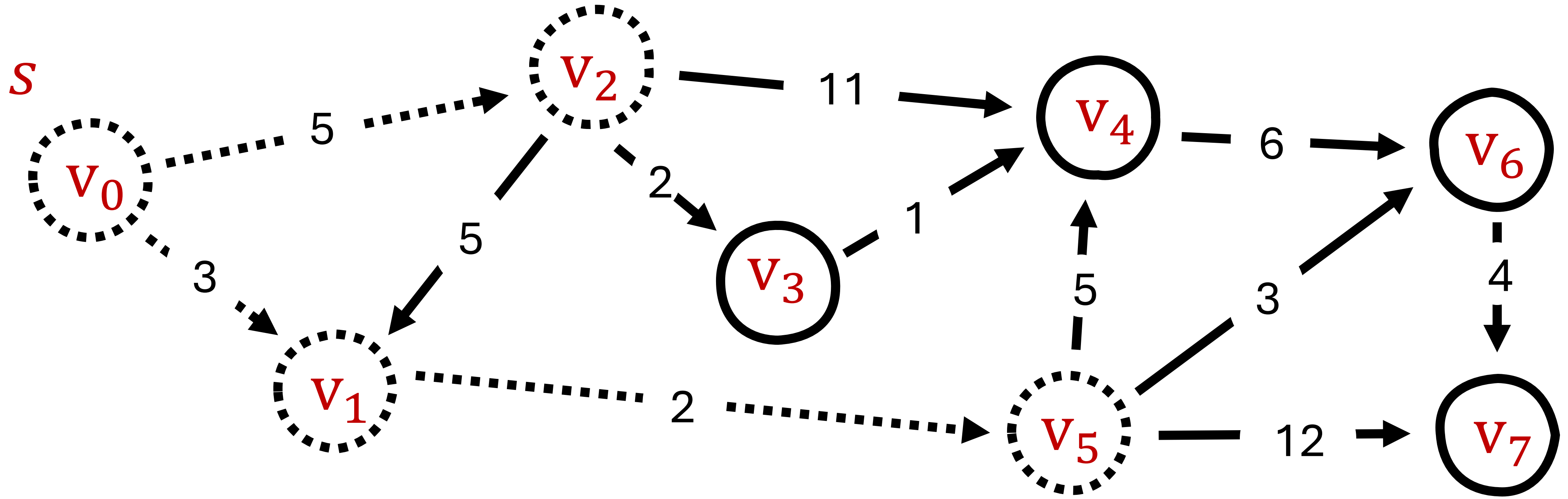
```
Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each u ∈ S, we store a distance d(u)
Initially S = {s} and d(s) = 0
While S ≠ V
    Select a node v ∉ S with at least one edge from S for which
       d'(v) = min_{e=(u,v):u∈S} d(u) + ℓ_e is as small as possible
    Add v to S and define d(v) = d'(v)
EndWhile
```



Dijkstra's Algorithm

S = {s}
d = [0, 3, 5, ∞, ∞, 5, ∞, ∞]

Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
  For each $u \in S$, we store a distance $d(u)$
Initially $S = \{s\}$ and $d(s) = 0$
While $S \neq V$
  Select a node $v \notin S$ with at least one edge from S for which
    $d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible
  Add v to S and define $d(v) = d'(v)$
EndWhile



Dijkstra's Algorithm

S = {s}

d = [0, 3, 5, 7, ∞, 5, ∞, ∞]

Dijkstra's Algorithm (G, ℓ)

Let S be the set of explored nodes

For each $u \in S$, we store a distance $d(u)$

Initially $S = \{s\}$ and $d(s) = 0$

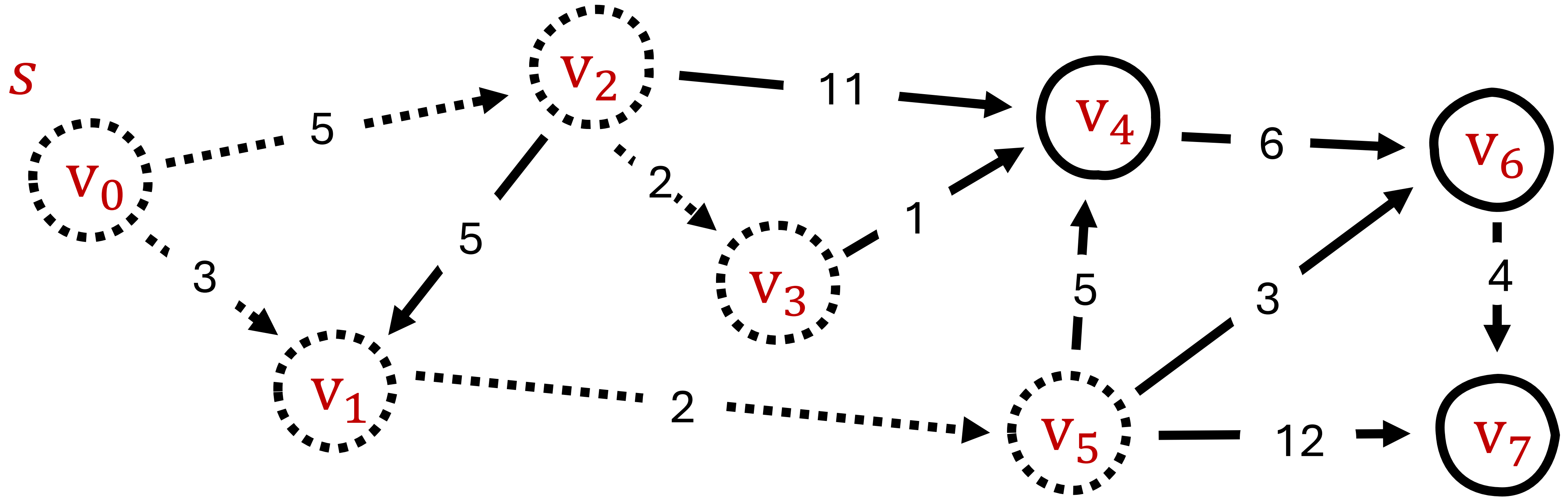
While $S \neq V$

Select a node $v \notin S$ with at least one edge from S for which

$d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible

Add v to S and define $d(v) = d'(v)$

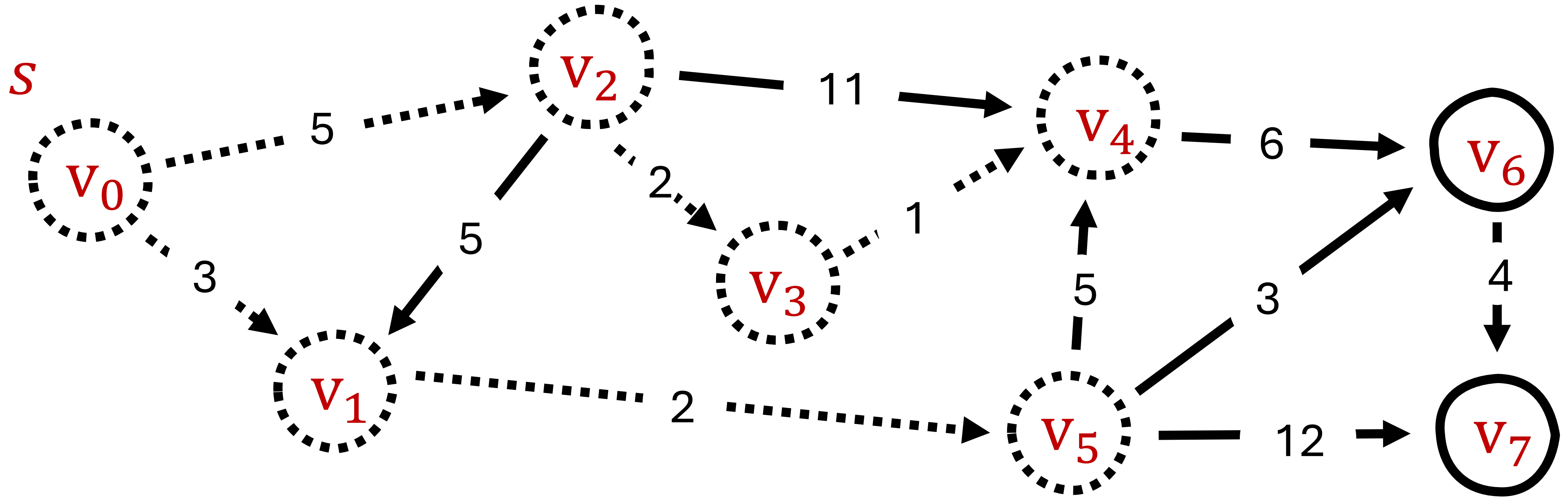
EndWhile



Dijkstra's Algorithm

S = {s}
d = [0, 3, 5, 7, 8, 5, ∞, ∞]

```
Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each u ∈ S, we store a distance d(u)
Initially S = {s} and d(s) = 0
While S ≠ V
    Select a node v ∉ S with at least one edge from S for which
        d'(v) = min_{e=(u,v):u∈S} d(u) + ℓ_e is as small as possible
    Add v to S and define d(v) = d'(v)
EndWhile
```

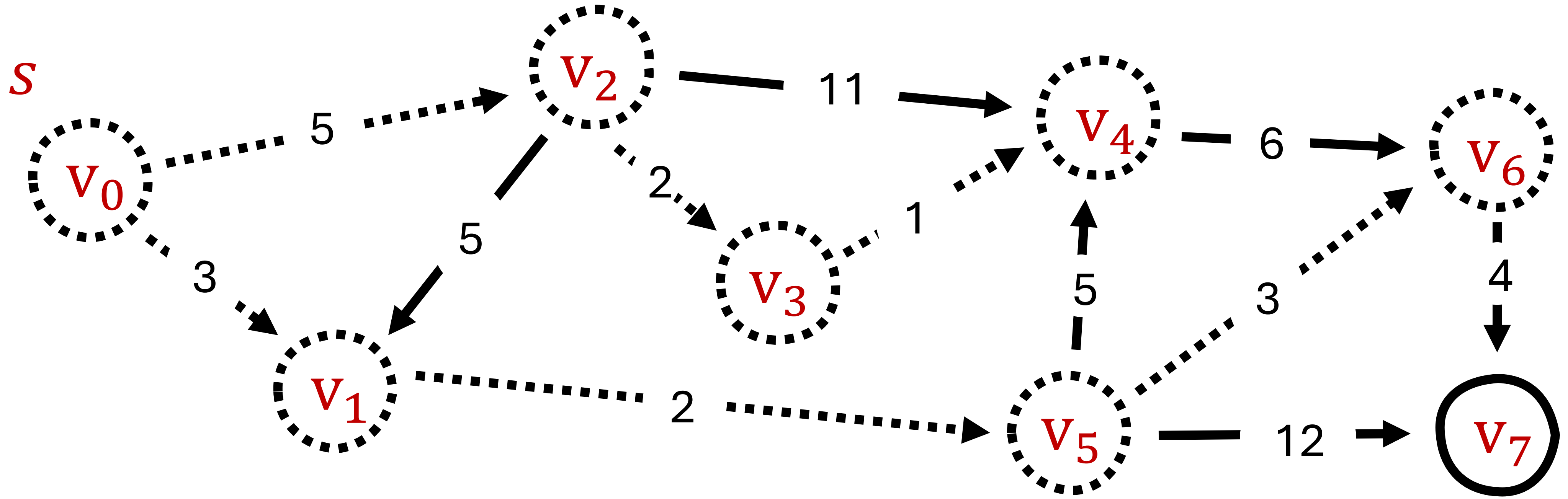


Dijkstra's Algorithm

S = {s}

d = [0, 3, 5, 7, 8, 5, 8, ∞]

```
Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each u ∈ S, we store a distance d(u)
Initially S = {s} and d(s) = 0
While S ≠ V
    Select a node v ∉ S with at least one edge from S for which
      d'(v) = min_{e=(u,v):u∈S} d(u) + ℓ_e is as small as possible
    Add v to S and define d(v) = d'(v)
EndWhile
```



Dijkstra’s Algorithm

S = {s}

d = [0, 3, 5, 7, 8, 5, 8, 12]

Dijkstra's Algorithm (G, ℓ)

Let S be the set of explored nodes

For each $u \in S$, we store a distance $d(u)$

Initially $S = \{s\}$ and $d(s) = 0$

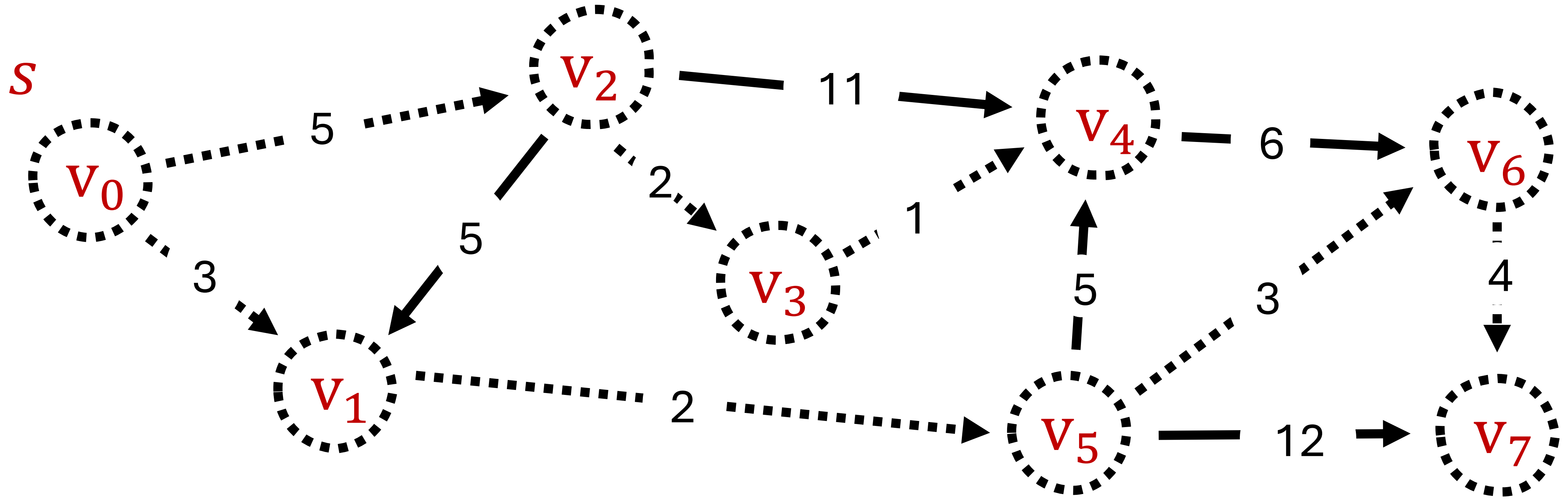
While $S \neq V$

Select a node $v \notin S$ with at least one edge from S for which

$d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible

Add v to S and define $d(v) = d'(v)$

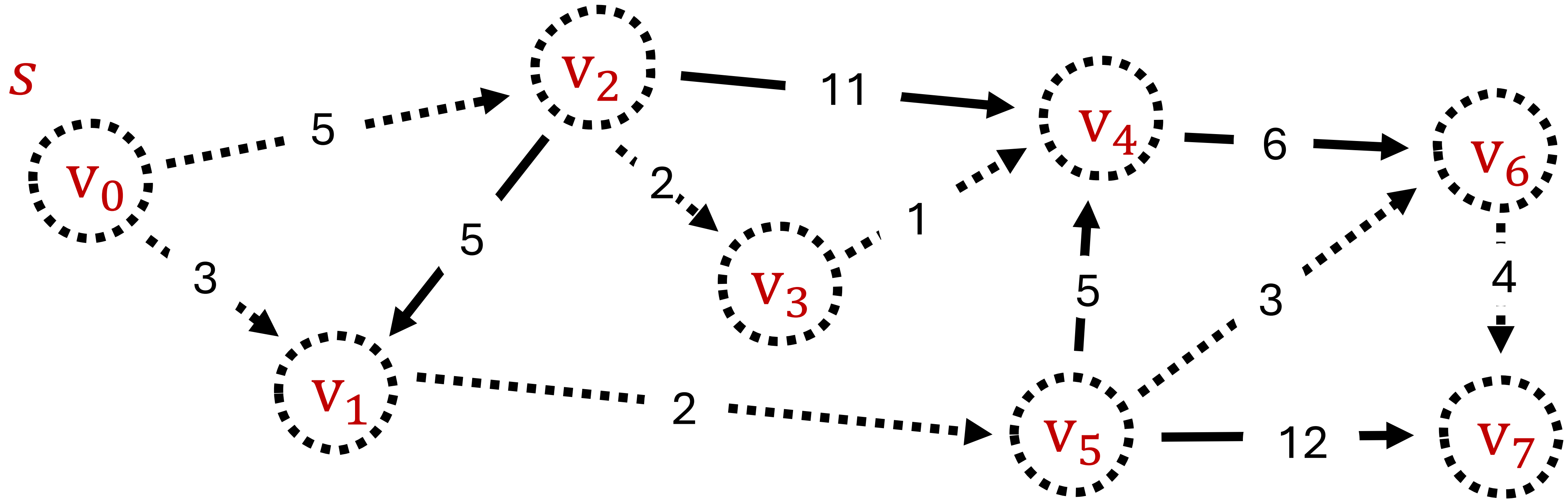
EndWhile



Paths?

Q: How do we return the paths?

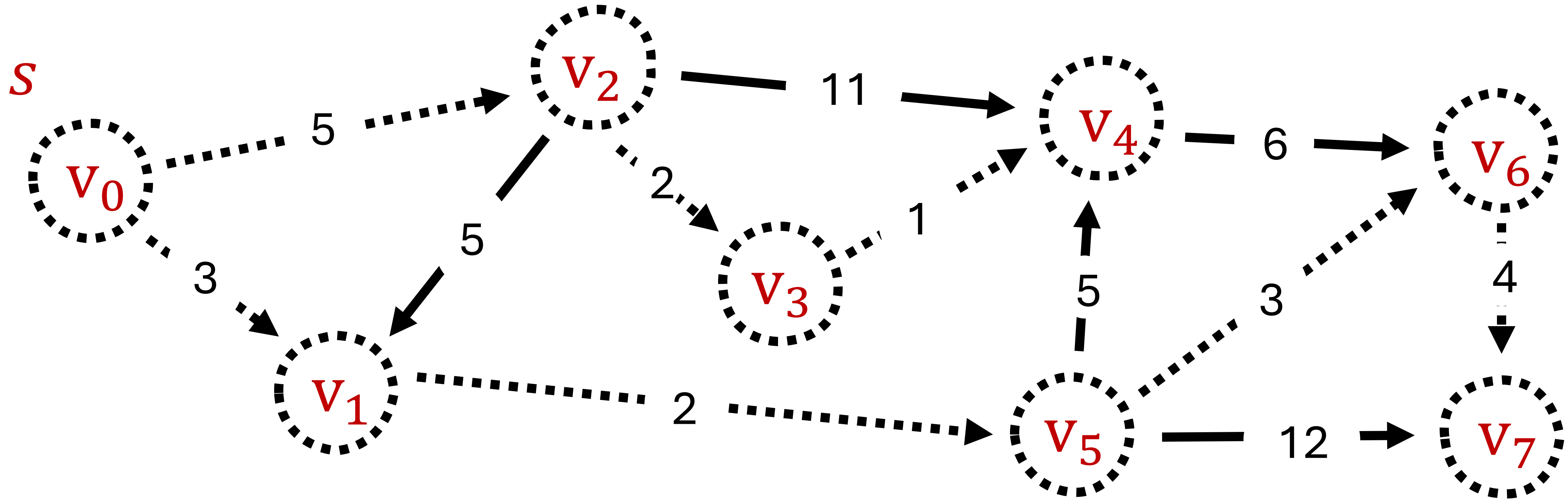
Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each $u \in S$, we store a distance $d(u)$
Initially $S = \{s\}$ and $d(s) = 0$
While $S \neq V$
    Select a node $v \notin S$ with at least one edge from S for which
        $d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible
    Add v to S and define $d(v) = d'(v)$
EndWhile



Paths?

Q: How do we return the paths?

```
Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each u ∈ S, we store a distance d(u)
Initially S = {s} and d(s) = 0
While S ≠ V
    Select a node v ∉ S with at least one edge from S for which
       d'(v) = min_{e=(u,v):u∈S} d(u) + ℓ_e is as small as possible
    Add v to S and define d(v) = d'(v)
EndWhile
```



Paths?

A: When we add v to S, we can also keep track of what edge was used.

Dijkstra's Algorithm (G, ℓ)

Let S be the set of explored nodes Let P a n length array.

For each $u \in S$, we store a distance $d(u)$

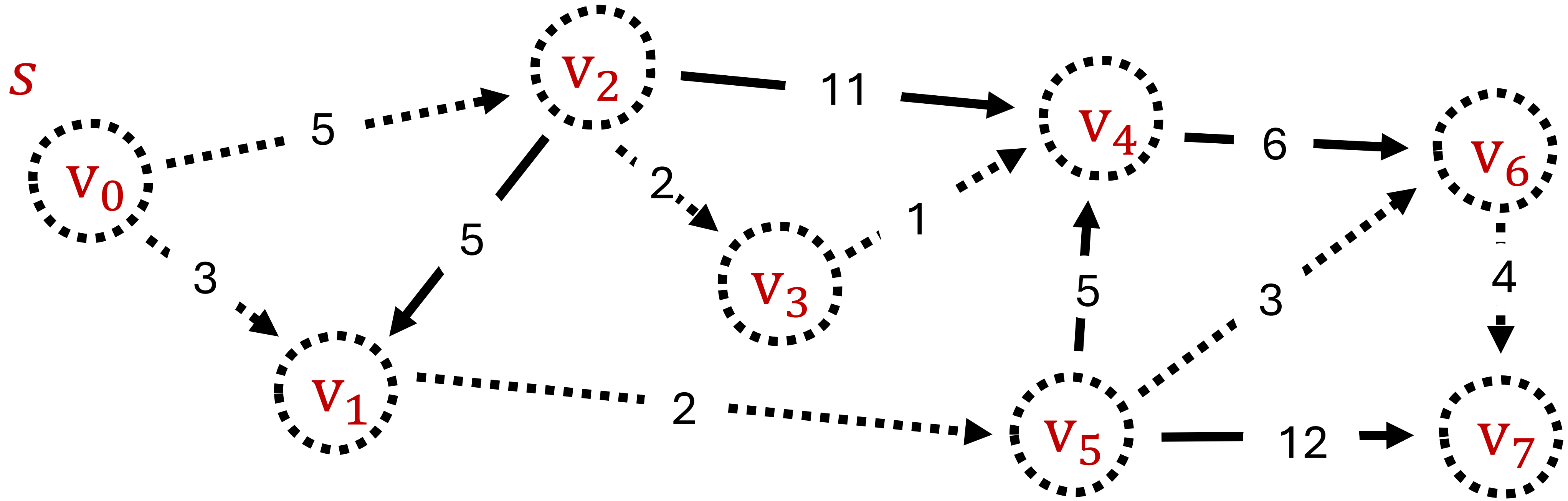
Initially $S = \{s\}$ and $d(s) = 0$ Set P[s] to be -1.

While $S \neq V$

Select a node $v \notin S$ with at least one edge from S for which $d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible

Add v to S and define $d(v) = d'(v)$ Set P[v] = u such that (u,v) was min.

EndWhile



Paths?

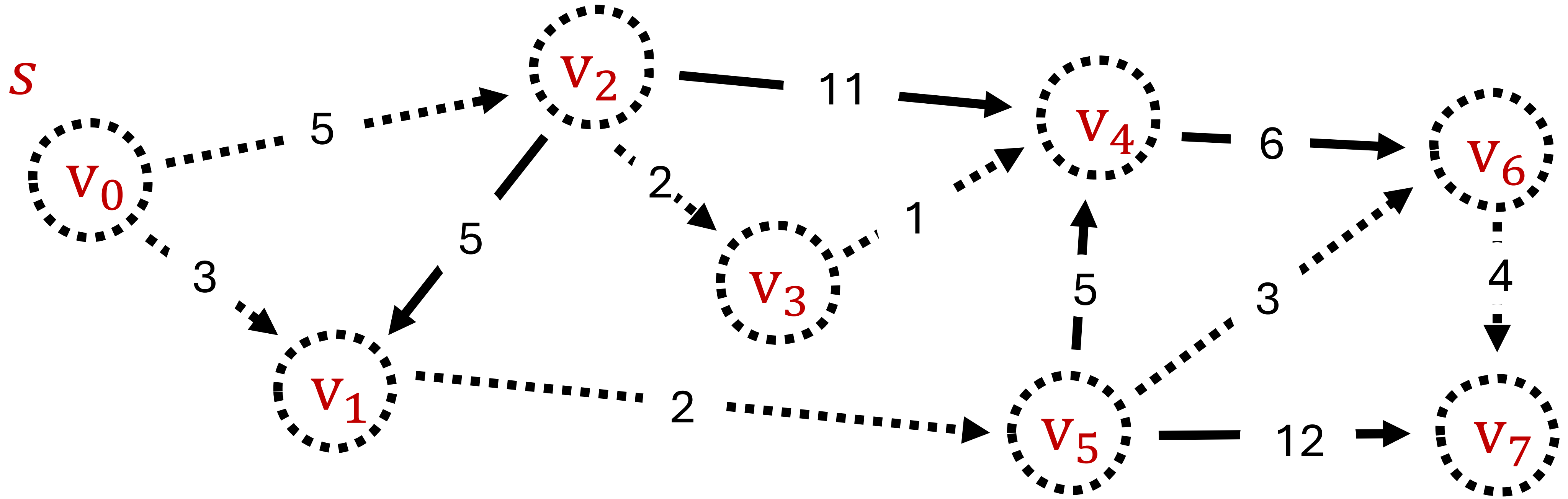
Q: How do we get the shortest path from s to w?

```
Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each u ∈ S, we store a distance d(u)
Initially S = {s} and d(s) = 0
While S ≠ V
    Select a node v ∉ S with at least one edge from S for which
        d'(v) = min_{e=(u,v):u∈S} d(u) + ℓ_e is as small as possible
    Add v to S and define d(v) = d'(v)
EndWhile
```

Let P a n length array.

Set P[s] to be -1.

Set P[v] = u such that (u,v) was min.



Paths?

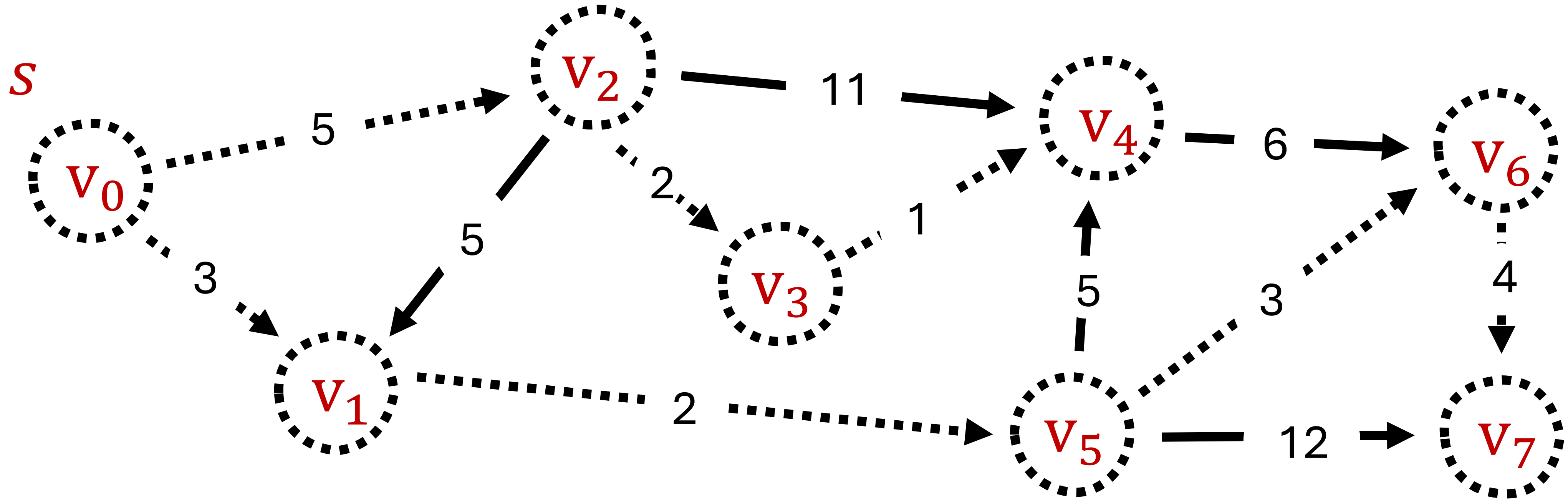
A: We can recursively construct it by looking at taking the edge (P[w],w) and the shortest path from s to P[w].

```
Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each u ∈ S, we store a distance d(u)
Initially S = {s} and d(s) = 0
While S ≠ V
    Select a node v ∉ S with at least one edge from S for which
        d'(v) = min_{e=(u,v):u∈S} d(u) + ℓ_e is as small as possible
    Add v to S and define d(v) = d'(v)
EndWhile
```

Let P a n length array.

Set P[s] to be Null.

Set P[v] = u such that (u,v) was min.



Paths

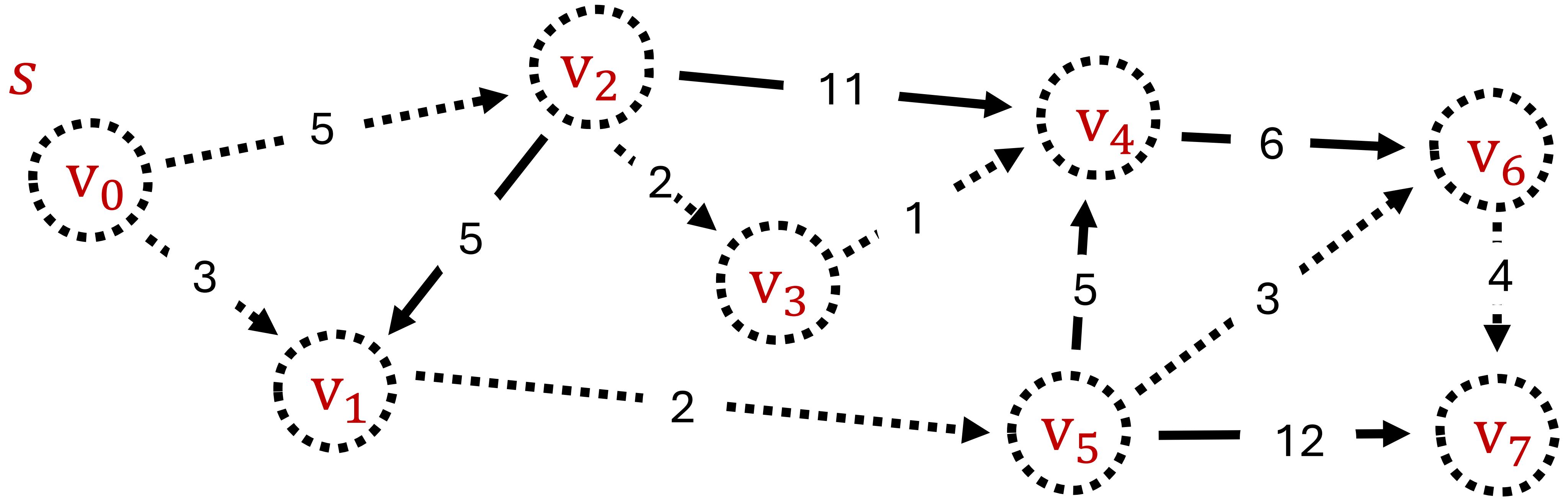
Let P_u be the path found from s to u using these modifications.

```
Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each u ∈ S, we store a distance d(u)
Initially S = {s} and d(s) = 0
While S ≠ V
    Select a node v ∉ S with at least one edge from S for which
        d'(v) = min_{e=(u,v):u∈S} d(u) + ℓ_e is as small as possible
    Add v to S and define d(v) = d'(v)
EndWhile
```

Let P a n length array.

Set P[s] to be -1.

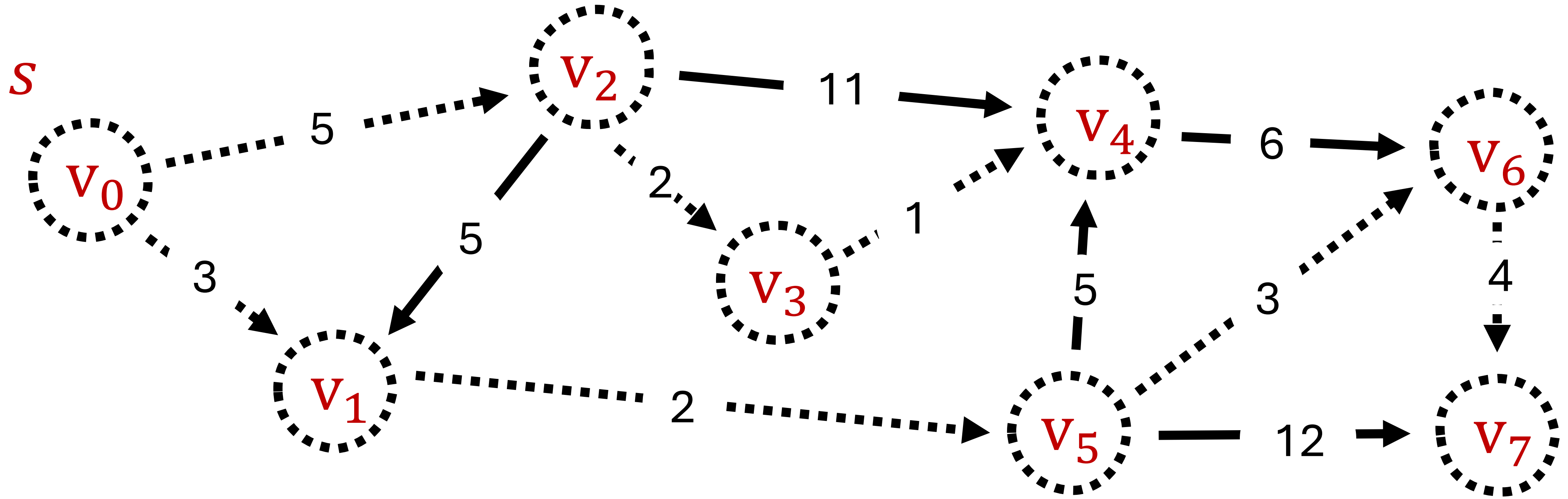
Set P[v] = u such that (u,v) was min.



Correctness

Claim: At the start of each loop, P_u is the shortest path from s to u for all u ∈ S.

```
Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each u ∈ S, we store a distance d(u)
Initially S = {s} and d(s) = 0
While S ≠ V
    Select a node v ∉ S with at least one edge from S for which
       d'(v) = min_{e=(u,v):u∈S} d(u) + ℓ_e is as small as possible
    Add v to S and define d(v) = d'(v)
EndWhile
```

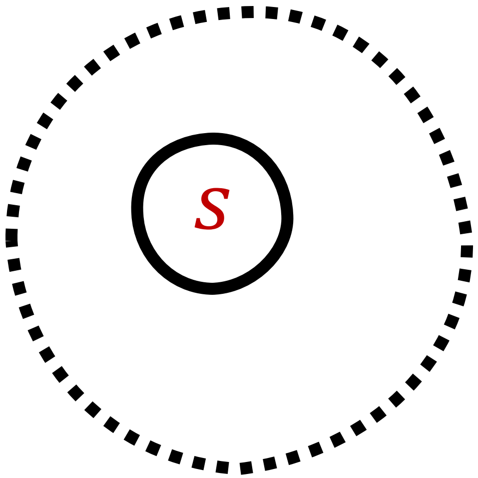


Proof Idea

- We proceed with induction on the size of S.
- If $|S| = 1$, then…

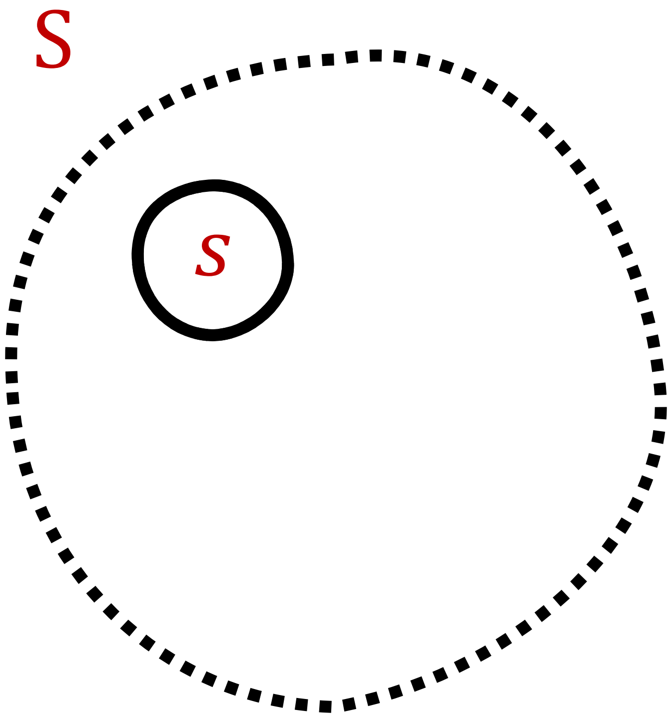
Proof Idea

- We proceed with induction on the size of S.
- If $|S| = 1$, then it is true because the shortest path from s to s is empty set.



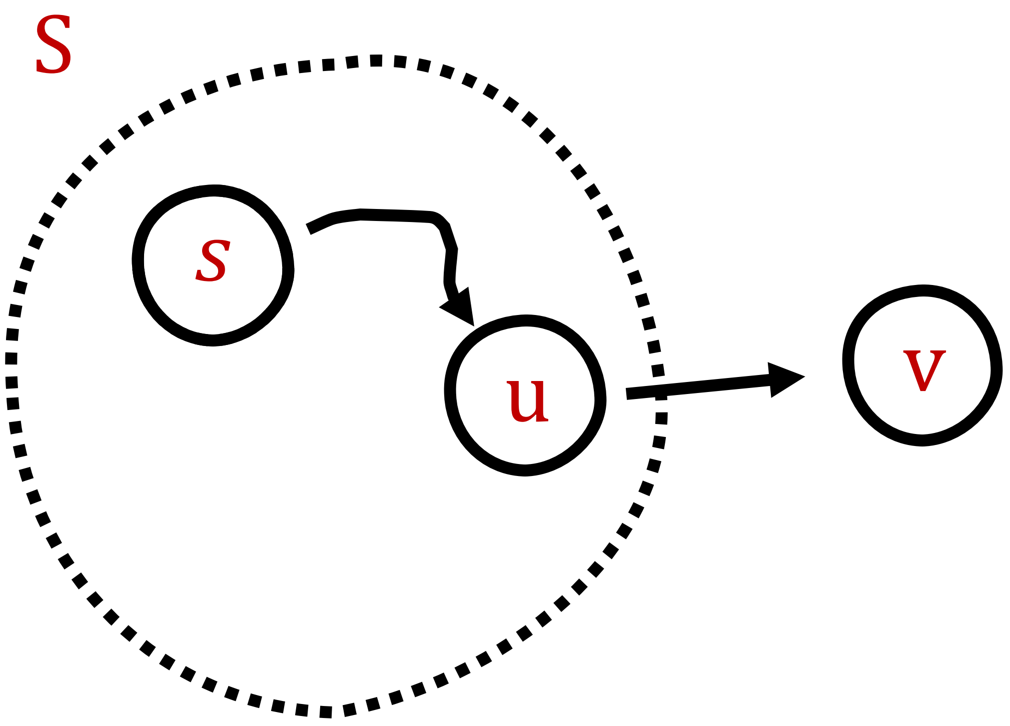
Proof Idea

- We proceed with induction on the size of S.
- Suppose for all S such that $|S| \leq \mathrm{k}$, P_u is the shortest path from s to u for all $\mathrm{u} \in \mathrm{S}$.



Proof Idea

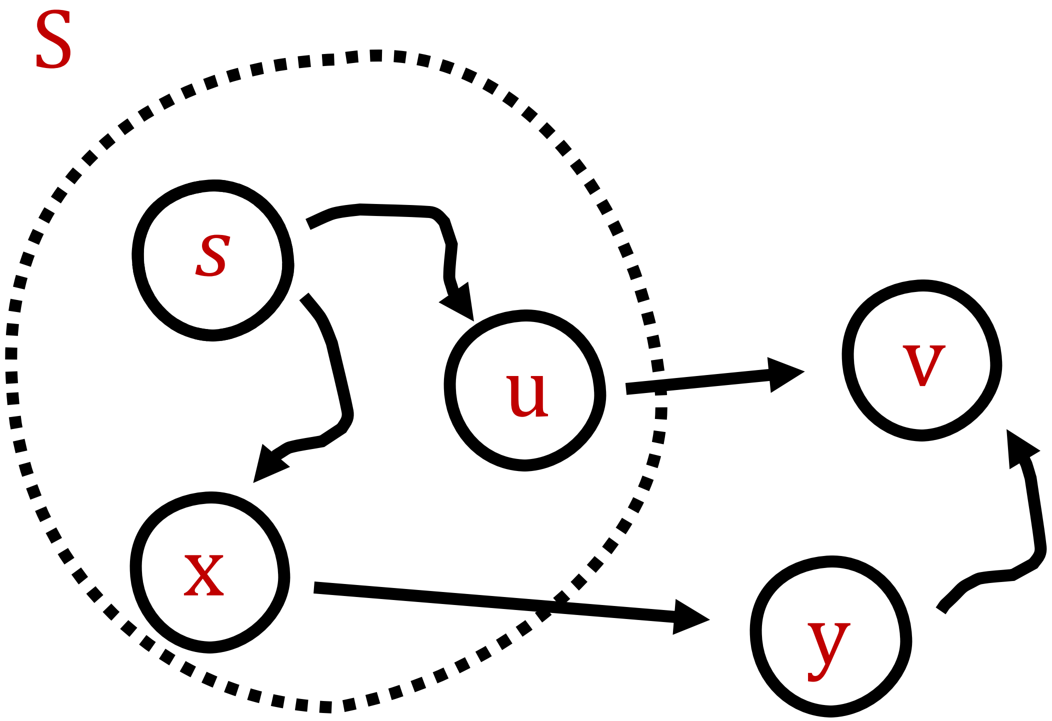
- We proceed with induction on the size of S.
- Suppose for all S such that $|S| \leq \mathrm{k}$, P_u is the shortest path from s to u for all $\mathrm{u} \in \mathrm{S}$.



- Let v be the k+1 vertex added to S and let (u,v) be the edge minimized $min_{e=(u,v):u \in S} d(u) + \ell_e$.

Proof Idea

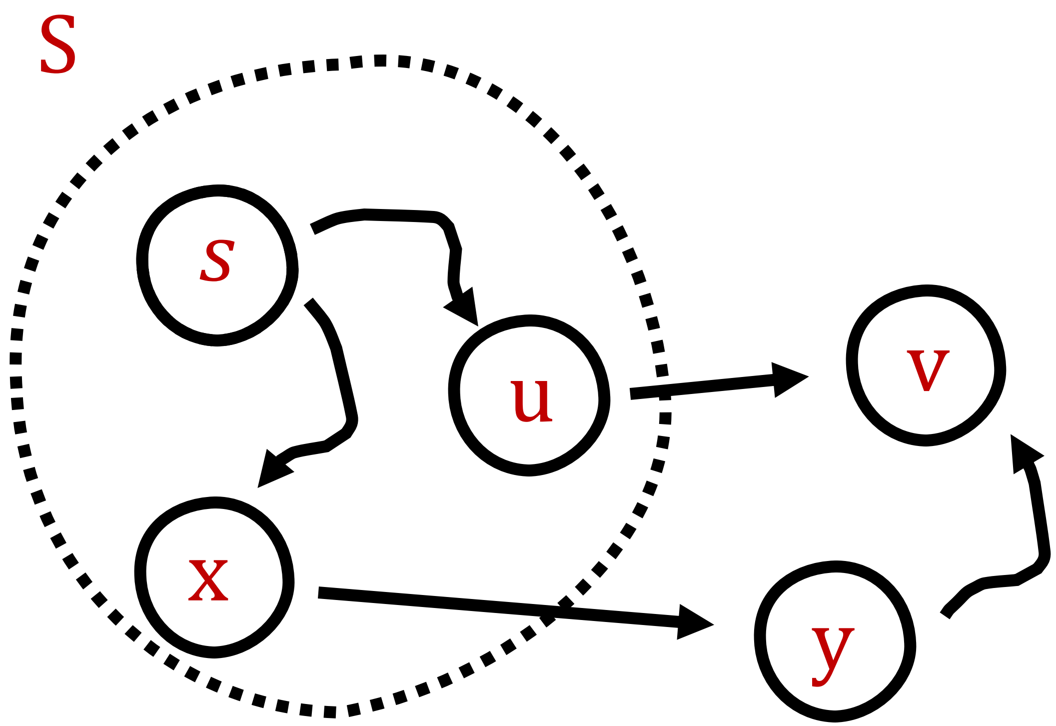
- We proceed with induction on the size of S.
- Suppose for all S such that $|S| \leq$ k, P_u is the shortest path from s to u for all u ∈ S.



- Let v be the k+1 vertex added to S and let (u,v) be the edge minimized $min_{e=(u,v):u \in S} d(u) + \ell_e$.
- Suppose $P_u \cup \{(u, v)\}$ is not the shortest path. Then there exists a path that must leave S via some edge (x,y).

Proof Idea

- We proceed with induction on the size of S.
- Suppose for all S such that $|S| \le \mathrm{k}$, P_u is the shortest path from s to u for all $\mathrm{u} \in \mathrm{S}$.



- Let v be the k+1 vertex added to S and let (u,v) be the edge minimized $min_{e=(u,v):u\in S} d(u) + \ell_e$.
- Suppose $P_u \cup \{(\mathrm{u},\mathrm{v})\}$ is not the shortest path. Then there exists a path that must leave S via some edge (x,y).
- However, the algo picked v and so the path from s to y must be just as long. ><

Q: What about negative edge weights?